Delft University of Technology
Parallel and Distributed Systems Report Series

# Memory Access Patterns on Architectures with Local Memory: A Performance Database

**Jianbin Fang, Ana Lucia Varbanescu**
{j.fang,a.l.varbanescu}@tudelft.nl

Completed in October 2012

**Report number PDS-2012-002**

TU Delft PDS

**ISSN 1387-2109**

Published and produced by:
Parallel and Distributed Systems Group
Department of Software and Computer Technology
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology
Mekelweg 4
2628 CD Delft
The Netherlands

Information about Parallel and Distributed Systems Report Series:
reports@pds.ewi.tudelft.nl

Information about Parallel and Distributed Systems Group:
http://www.pds.ewi.tudelft.nl/



J. Fang, A.L. Varbanescu

Memory Access Patterns on Architectures with LM

**Abstract**

Nowadays architectures and their programming model implementations are becoming increasingly complex and diverse, making the performance benefits of using local memory unpredictable via only simplistic modeling. In this paper, we present a benchmark-based approach to tackle this issue. We first present a two-part approach to describe memory access patterns for many-thread applications. For each MAP, we design benchmarks of native versions (without local memory) and optimized versions (using local memory). Then we evaluate them on typically used platforms (NVIDIA GTX280, NVIDIA GTX580, AMD HD6970, and Intel E5620), compare the performance of native versions versus optimized versions, and get a performance database. This database can provide essential information for automated usage of local memory.

**Keywords:** Performance Measurement, Benchmarking, Local Memory, Memory Access Patterns.

# Contents

# List of Figures



# List of Tables

# 1 Introduction

In the last few years, multicore/manycore processors have become increasingly popular. To exploit the full benefits of the increasing number of computational units, architects need to ensure that memory bandwidth and latency are optimized [1]. Utilizing a cache hierarchy has been the traditional way to alleviate the memory bottleneck [2], but there are still various modern parallel architectures such as NVIDIA Fermi and AMD Northern Island that use programmer-managed scratch-pad memory, referred to as *local memory* [1].

Because local memory is situated on-chip, it is much faster than the global memory (e.g., reading data from local memory is 3× faster on NVIDIA Quadro5000 and 5× faster on AMD Radeon HD6970). Thus, a proper use of local memory often leads to higher memory bandwidth and thus performance improvement. Nevertheless, using local memory is an error-prone and time-consuming process. Programmers often have to manually address, in their code, challenges like (1) geometry mismatch, (2) work-items masking and binding switches, and (3) inefficient local memory organization [3]. We argue that when solving these problems manually, programmers waste too much time on non-computational and non-functional coding details, which hinders productivity and bloats the code.

To tackle these challenges, the ultimate solution is to use automated code transformation, typically in the form of a compiler pass- *given an OpenCL kernel without usage of local memory, the compiler pass translates the input kernel to a format with local memory.* Thus, local memory becomes an oblivious feature from programmers' point of view. Generally, this code translation consists of two steps: (1) predict the performance benefits of using local memory, and then (2) perform code translation [1]. The first step provides essential information on whether using local memory will produce performance gain for the second step and dictates the necessity (or otherwise lack thereof) to run the second step.

Regarding the first step, *data reuse* is a commonly recognized performance metric of performance benefits, i.e., threads sharing data elements leads to memory bandwidth improvement [4]. Thus, predicting performance benefits could be as easy as identifying the existence of data reuse. But, we have found that the assumption that when there is data reuse, there are performance benefits from using local memory, does not always hold (see Section 2.1). On the other hand, without data reuse, we can still achieve bandwidth increase by using local memory to change the access patterns of global memory, e.g., on GPUs (see Section 2.2). Furthermore, the architecture diversity makes performance prediction even more complex. For example, most GPUs are featured with on-chip local memory, while current CPUs emulate local memory as a part of global memory space. Therefore, programmers often choose not to use local memory on such platforms (i.e., CPUs) to avoid any performance loss [5]. We have proved that properly using local memory on CPUs can enable performance improvement (see Section 2.3). As a result, the architecture diversity and implementation differences generate many counter-intuitive cases, making the performance benefits of using local memory on various architectures unpredictable.

In this paper, we focus on *addressing the issue of unpredictability of performance benefits from using local memory.* To do so, we develop a microbenchmark-based approach to quantify the performance impacts of using local memory. To ensure completeness, we design our benchmark based on *memory access patterns (MAP).* Benchmarks for each MAP include at least two cases: (1) the implementation without local memory (i.e., native kernels using global memory only), and (2) implementations using local memory. Then we evaluate the benchmarks on typically used platforms, and record the results (i.e., memory bandwidth) in a *performance database.*

Summarizing, this paper presents:

- An approach to describe memory access patterns (MAP) (see Section 3).
- A set of self-designed microbenchmarks with and without local memory in OpenCL (see Section 4).

[1] NVIDIA uses the term 'shared memory', while AMD calls it 'local data store'. In this paper, we will use the name 'local memory' (and other terms) from OpenCL.

- A performance database of running the microbenchmarks on four typically used platforms (see Section 5).
- A use-case for the performance database (see Section 6).

# 2 Motivation

Our work is based on the following observations and analysis.

## 2.1 Data Reuse ≠ Performance Improvement

The occurrence of data reuse is the widely used criterion of moving data from global memory to local memory. However, this assumption does not always hold. Table 1 shows the memory bandwidth when running NBody on NVIDIA GTX580. We see that using local memory performs worse than not using it (by around 20%). Data elements of the input are shared by all the threads, i.e., *full* data reuse. The performance loss is due to the fact that caches (L1 and L2) make better use of data sharing than the local memory. Specifically, local memory enables data sharing among work-items within one work-group, while the L1 cache can identify the data sharing within one work-group, and the L2 cache will enable global data sharing on the input data (i.e., among work-groups as well). Additionally, using local memory introduces extra overheads for data movement operations in and out of local memory. Therefore, the caches make quite an impact on the overall performance of applications like NBody.

Table 1: Memory bandwidth (GB/s) of NBody where the tile size is obtained dynamically ($LM_{w/o}$ represents the native kernels, and $LM_{w/i}$ represents the kernels using local memory).

| | 64x64 | 128x128 | 256x256 | 512x512 | 1024x1024 |
|---|---|---|---|---|---|
| $LM_{w/o}$ | 613.50 | 636.43 | 646.06 | 616.42 | 589.95 |
| $LM_{w/i}$ | 512.44 | 495.28 | 516.04 | 518.61 | 520.64 |

## 2.2 No Data Reuse ≠ Performance Loss

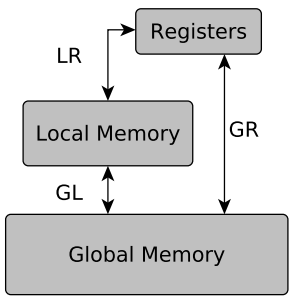



Figure 1: Logical view of architectures with local memory.

Figure 1 gives a logical view of data movements among registers, local memory, and global memory. Suppose we have $N$ compute units and the bandwidth of local memory access is $W_l$. An application requires $D$ data

elements to be moved when using global memory only (with a bandwidth of $W_g$), and $D^{'}$ data elements to be moved from global memory to local memory (with a bandwidth of $W_g^{'}$). We compute the time of data movement with ($T_{w/i}$) and without local memory ($T_{w/o}$) as follows:

$$T_{w/o} = \frac{D}{W_g} \tag{1}$$

$$T_{w/i} = \begin{cases} \frac{D}{W_l} + \frac{D^{'}}{W_g^{'}} & N = 1 \\ max(\frac{D}{W_l}, \frac{D^{'}}{W_g^{'}}) = \frac{D^{'}}{W_g^{'}} & N > 1 \end{cases} \tag{2}$$

We see that performance improvement comes from two factors: either the decrease of data amount ($D^{'} < D$), and/or the increase of global memory bandwidth ($W_g^{'} > W_g$). Thus, the assumption of considering data reuse as a must for performance gain does not hold. Taking Matrix Transpose on GPUs for example, we violate the coalescing access constraints on the global memory access. With local memory, we can ensure coalescing when loading data from global memory, and thus $W_g^{'} > W_g$.

### 2.3 Using Local Memory on CPUs ≠ Performance Loss

At the moment of writing, local memory is allocated on global memory space on CPUs. Thus, it is not recommended to use local memory on CPUs [5]. However, we have found that this does not always hold. Table 2 shows the memory bandwidth of a convolution kernel on the Intel E5620 processor. We see that using local memory delivers better performance than not using it (around 2× faster). Using local memory on CPUs introduces extra overheads, but it also changes the usage of caches, i.e., it transforms the usage of one single large memory to multiple small memory spaces localized per core, thus possibly leading to better utilization of caches.

Table 2: Memory bandwidth (GB/s) of convolution with and without local memory.

| | 64x64 | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 |
|---|---|---|---|---|---|---|
| $LM_{w/o}$ | 6.81 | 7.77 | 7.81 | 8.06 | 8.13 | 8.15 |
| $LM_{w/i}$ | 12.23 | 13.81 | 14.08 | 14.56 | 14.70 | 14.56 |

Based on these observations, we believe that it is intricate to predict the performance benefits of using local memory only via intuitive understanding and simplistic modeling. Thus, we propose a microbenchmarking approach to tackle this issue.

## 3 MAP Description

In this section, we present a mathematical model that captures the memory access pattern in an OpenCL kernel. We extend the model based on prior work that targets vectorization and memory selection [6].

The memory access sequence $\vec{s}$ is decomposed into two parts:

$$\vec{s} = M\vec{i} + \vec{o}, \tag{3}$$

where the first part $M\vec{i}$ represents the interthread memory access patterns *(eMAP)*, and the second part $\vec{o}$ represents the intrathread access patterns *(iMAP)*. eMAP generates a base access index for the thread $(ty,\ tx)$, while iMAP provides an offset which represents the distance from the base address.

Suppose that we have a 2D matrix and a 2D thread configuration (Higher dimensions can be seen as multiple 2D dimensions, and 1D is a special case of 2D). Thus, the access sequence is now simplified as:

$$\vec{s} = \begin{bmatrix} M_{00} & M_{01} \\ M_{10} & M_{11} \end{bmatrix} \begin{bmatrix} t_y \\ t_x \end{bmatrix} + \begin{bmatrix} o_0 \\ o_1 \end{bmatrix} \tag{4}$$

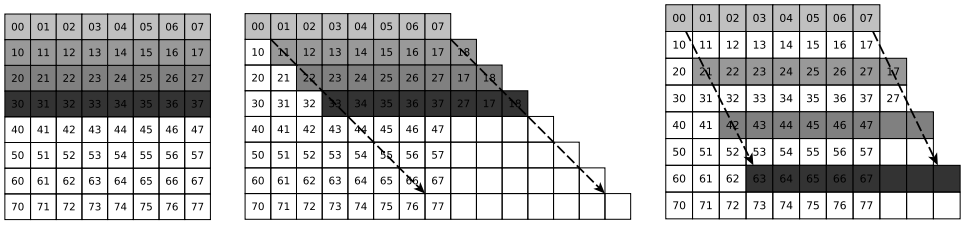
## 3.1 eMAP

When $M_{00}, M_{01}, M_{10}, M_{11} \in \{0, 1\}$, we generate 16 cases of eMAP (shown in Figure 2).

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

Figure 2: eMAP cases (they are numbered starting with 01 til 16 in the left-to-right and top-down order).

As we have mentioned, eMAP provides the base index of memory reference. For example, the base address of eMAP-14 (the one shaded in Figure 2) for each thread is shown in Figure 3(b). We see that the work-items in the x dimension of a work-group access continuous data elements lying in the horizontal direction; the work-items in the y dimension of a work-group reference continuous data elements lying on the diagonal line.



(a) Work-group (b) eMAP-14 Base Address (c) Base Address when $M_{00} = 2$

Figure 3: Work-group and the base address.

When $M_{00}, M_{01}, M_{10}, M_{11} \notin \{0, 1\}$, the eMAPs become more complex. Figure 3.1 shows the case when $M_{00} = 2$. We see (from Figure 3(c)) that there will be 'gaps' between rows due to the larger stride. We can imagine that any non-unit stride will introdue 'holes' or 'gaps' for a general case. For now, we only consider the most commonly used eMAP cases (in Figure 2) in our work. We believe the extension to larger strides is not too complex, but it will lead to cases very rarely seen in real applications. This step is left for future work.

$$\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \Longrightarrow \begin{bmatrix} 2 & 0 \\ 1 & 1 \end{bmatrix}$$

Figure 4: The case when $M_{00} = 2$

## 3.2 iMAP

iMAP captures the memory access patterns of a single thread. The commonly typical iMAPs (in real-life applications) are Single, Row, Column, Block, and Neighbor:

- Single: each thread accesses only one data element indexed by the base address.
- Row: each thread references a row of data elements within the row indexed by the base address.
- Column: each thread accesses a column of data elements within the column indexed by the base address.
- Block: each thread accesses a block of data elements within the block centered at the base address and of radius R.
- Neighbor: each thread access the data element lying at the base address and its four neighbors.

The iMAP representations are listed as follows:

$$Single: \ \vec{o}_k = O_{(k)}, O = \left\{ \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right\}$$

$$Row: \ \vec{o}_k = O_{(k)}, O = \left\{ \begin{bmatrix} 0 \\ i \end{bmatrix} \ | \ 0 \le i < W, i \in N \right\}$$

$$Column: \ \vec{o}_k = O_{(k)}, O = \left\{ \begin{bmatrix} j \\ 0 \end{bmatrix} \ | \ 0 \le j < H, j \in N \right\}$$

$$Block: \ \vec{o}_k = O_{(k)}, O = \left\{ \begin{bmatrix} j \\ i \end{bmatrix} \ | \ 0 \le i < X, i \in N; \ 0 \le j < Y, j \in N \right\}$$

$$Neighbor: \ \vec{o}_k = O_{(k)}, O = \left\{ \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right\}$$

## 3.3 Put it together

Once eMAP and iMAP are specified, we will get 80 ($16 \times 5$) memory access patterns, and thus need to implement 80 microbenchmarks. However, we find that each of the iMAP has its access constraints on either of the two dimensions, e.g., the Single iMAP requires that its eMAP specify an index for both high and low dimension for the data matrix, while the Row iMAP needs only the high dimension index. The combination constraints for each iMAP are listed in Table 3.

Table 3: Dimension specification for each iMAP ('Y' means we need to specify the index for the current dimension, 'N' represents we need not to specify the current index, and 'Y/N' represents we need to specify both Y or both N).

| | **H** | **L** |
|---|---|---|
| **Single** | Y | Y |
| **Row** | Y | N |
| **Column** | N | Y |
| **Block** | Y/N | Y/N |
| **Neighbor** | Y | Y |

The reduced space of memory access patterns (i.e., taking into account Table 3) is shown in Table 4. We see that only 34 valid MAPs are generated. For each of them, we design benchmarks, as presented in Section 4.

Table 4: All the memory access patterns (N/A represents the combination does not apply here).

| | 1 Single | 2 Row | 3 Column | 4 Block | 5 Neighbor |
|---|---|---|---|---|---|
| **01** | N/A | N/A | N/A | 401 | N/A |
| **02** | N/A | N/A | 302 | N/A | N/A |
| **03** | N/A | N/A | 303 | N/A | N/A |
| **04** | N/A | 204 | N/A | N/A | N/A |
| **05** | N/A | 205 | N/A | N/A | N/A |
| **06** | N/A | N/A | 306 | N/A | N/A |
| **07** | 107 | N/A | N/A | 407 | 507 |
| **08** | 108 | N/A | N/A | 408 | 508 |
| **09** | 109 | N/A | N/A | 409 | 509 |
| **10** | 110 | N/A | N/A | 410 | 510 |
| **11** | N/A | 211 | N/A | N/A | N/A |
| **12** | 112 | N/A | N/A | 412 | 512 |
| **13** | 113 | N/A | N/A | 413 | 513 |
| **14** | 114 | N/A | N/A | 414 | 514 |
| **15** | 115 | N/A | N/A | 415 | 515 |
| **16** | 116 | N/A | N/A | 416 | 516 |

# 4 Microbenchmarking

In this section, we explore the design space of implementations with and without local memory, and give implementations for each MAP.

## 4.1 Benchmarking Framework

Figure 5 shows the benchmarking framework. Given the MAPs from Section 3, we design benchmarks without (native kernels) and with local memory. Then we evaluate the benchmarks on typically used platforms and generate the performance database.

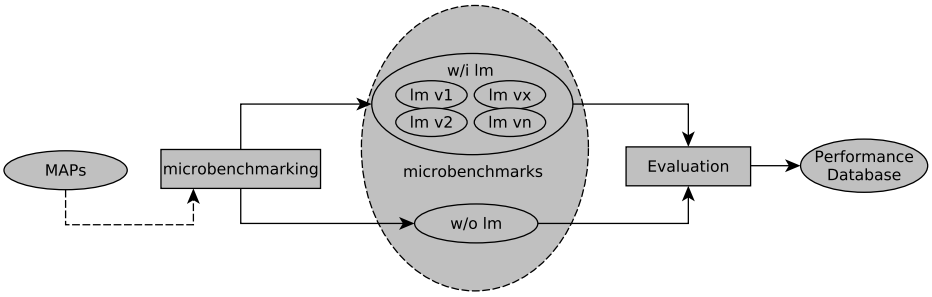



Figure 5: Benchmarking Framework.

## 4.2 Benchmark Design Space

Global memory access generally consists of two basic operations: (1) read (from global space) operations, and (2) write (into global space) operations. Compared with write operations, read operations are more complex, because for data parallel applications we usually decompose problems based on the output data. Further,

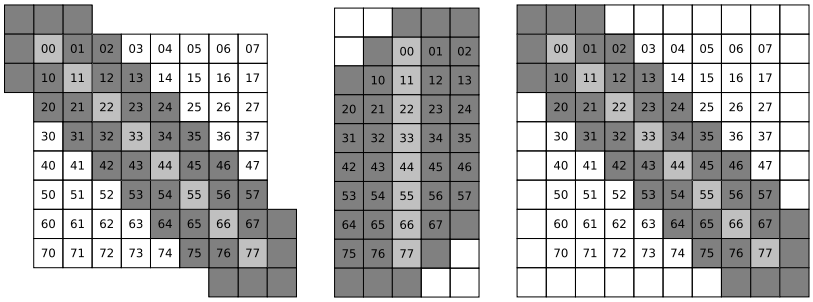
benchmarks for reading operations can be applied to writing operations by simply rewriting the benchmark. Therefore, our work focuses on read operations.

When designing a benchmark (for a MAP) with local memory, we need to consider the issue of *local space allocation* and *local memory access*.

### 4.2.1 Local Space Allocation

OpenCL provides programmers with two ways to allocate local memory space. The first is to *statically* allocate the space in kernels with predefined size, and the second way is to *dynamically* allocate the space in APIs during run-time. We prefer the former approach if the size can be known in advance, because the compiler may perform optimizations (e.g., loop unrolling) in this situation.

Regarding the size of local space, we prefer using a space of *moderate* size (M-approach), i.e., a local memory space to hold the necessary data elements with none or very few wasted cells. On the other hand, programmers may use a large enough space (L-approach). The two approaches are shown in Figure 6 for MAP-407 (R=1). The M-approach needs a local space of $(4R + 1) \times (S + 2R)$ data elements (where $R$ is the radius, and $S$ is the width or height of a work-group), while the L-approach needs $(S + 2R)^2$. Thus, the M-approach consumes less local memory, possibly enabling more work-groups to keep active simultaneously and better performance. However, programmers need to spend much more time mapping local work-items to local space and global work-items to global data elements.




(a) (b) M-approach (c) L-approach

Figure 6: Two approaches to load data elements for MAP-407 when R=1: (a) the shared data elements are required.

### 4.2.2 Local Memory Access

We need to ensure that the work-items within one work-group reference the data elements efficiently. A key optimization is to avoid bank conflicts, i.e., to ensure that access requirements from multiple work-items fall into different banks.

# 5 Performance Database

## 5.1 Performance Metric

We use memory bandwidth as the performance comparison metric. Suppose we have $W \times H$ threads, and each needs $N$ data elements of `type` datatype. We run each kernel $R$ times and measure the total execution time $T$.

Then we calculate the bandwidth, $B$(GB/s), as follows:

$$B = \frac{W \times H \times N \times sizeof(type)}{T/R} \times 10^{-9}$$

We measure the memory bandwidth for cases with and without local memory, use the number obtained without local memory as the reference, and calculate the speedup ($SP$) of using local memory. If $SP > 1$, using local memory performs better than the case without it; otherwise, using local memory leads a loss in memory bandwidth. Further, we classify the performance impacts into four groups: (1) PD (performance decrease), if $(SP - 1) < -5\%$; (2) PS (similar performance), if $|SP - 1| \leq 5\%$; (3) PI (performance increase), if $5\% < (SP - 1) \leq 100\%$; (4) PSI (significant increase), if $(SP - 1) > 100\%$ (shown in Figure 7).

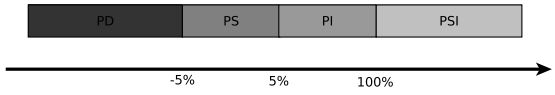



Figure 7: Performance impacts classification.

## 5.2 Experimental Setup

We run and compare the benchmarks on four platforms, of which configurations are shown in Table 5. In the experiment context, we use the device name GTX280, HD6970, GTX580, and E5620 to represent Platform I, Platform II, Platform III, and Platform IV, respectively.

Table 5: Details of underlying platforms

| | **Platform I** | **Platform II** | **Platform III** | **platform IV** |
|---|---|---|---|---|
| **Host** | Intel Core i7 920 | Intel Xeon E5620 | Intel Xeon E5620 | Intel Xeon E5620 |
| **Host OS** | UBUNTU v11.10 | CentOS v6.2 | CentOS v6.2 | CentOS v6.2 |
| **Device** | NVIDIA GTX280 | AMD HD6970 | NVIDIA GTX580 | Intel Xeon E5620 |
| **GCC version** | v4.6.1 | v4.4.6 | v4.4.6 | v4.4.6 |
| **OpenCL version** | CUDA v4.2 | AMD APP v2.7 | CUDA v4.2 | Intel OCL SDK v2.0 |

When measuring the performance, we use six data sets: $128 \times 128$, $256 \times 256$, $512 \times 512$, $1024 \times 1024$, $2048 \times 2048$, $4096 \times 4096$. 'w/o' represents the native kernels without local memory, 'w/i' represents the kernels using local memory, and 'sp' represents the speedup of 'w/i' compared with 'w/o'. For the MAPs-4**, we set the radius to be 3.

## 5.3 Performance Results

### 5.3.1 MAP-107

We see that using local memory significantly improves the memory bandwidth on GTX280, especially for large data sets. On HD6970 and GTX580, we also achieve some performance gains. The performance of kernels with and without local memory is similar on E5650. Using local memory for this MAP changes access patterns of global memory and enables data reuse for threads with the same $ty$ index.

Table 6: Performance impacts of using local memory for MAP-107

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 3.81 | 9.52 | 16.01 | 21.33 | 23.14 | 11.17 |
| w/i | 4.37 | 14.52 | 35.13 | 60.20 | 71.64 | 75.79 |
| sp | 1.15 | 1.53 | 2.19 | 2.82 | 3.10 | 6.79 |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 1.14 | 4.00 | 11.08 | 24.90 | 60.00 | 82.81 |
| w/i | 1.27 | 4.68 | 14.46 | 44.08 | 67.02 | 82.45 |
| sp | 1.12 | 1.17 | 1.31 | 1.77 | 1.12 | 1.00 |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 6.47 | 22.63 | 58.22 | 94.12 | 111.41 | 99.39 |
| w/i | 6.58 | 23.18 | 66.75 | 125.16 | 157.33 | 150.18 |
| sp | 1.02 | 1.02 | 1.15 | 1.33 | 1.41 | 1.51 |
| **E5620** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 2.41 | 6.48 | 12.29 | 16.36 | 18.39 | 15.36 |
| w/i | 2.71 | 6.87 | 11.87 | 15.56 | 17.66 | 15.92 |
| sp | 1.12 | 1.06 | 0.97 | 0.95 | 0.96 | 1.04 |

### 5.3.2 MAP-108

Table 7: Performance impacts of using local memory for MAP-108

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 4.35 | 14.54 | 39.70 | 76.88 | 96.92 | 104.68 |
| w/i | 4.30 | 13.88 | 34.22 | 61.77 | 74.32 | 77.74 |
| sp | 0.99 | 0.96 | 0.86 | 0.80 | 0.77 | 0.74 |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 1.11 | 3.67 | 7.98 | 22.25 | 61.28 | 82.87 |
| w/i | 1.28 | 4.72 | 15.68 | 44.44 | 65.58 | 82.96 |
| sp | 1.15 | 1.29 | 1.96 | 2.00 | 1.07 | 1.00 |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 6.72 | 24.48 | 64.23 | 115.02 | 145.94 | 154.62 |
| w/i | 6.55 | 23.88 | 61.26 | 108.53 | 137.06 | 145.32 |
| sp | 0.98 | 0.98 | 0.95 | 0.94 | 0.94 | 0.94 |
| **E5620** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 2.89 | 7.38 | 15.68 | 24.57 | 14.36 | 12.40 |
| w/i | 2.53 | 5.91 | 9.61 | 12.02 | 9.34 | 8.76 |
| sp | 0.88 | 0.80 | 0.61 | 0.49 | 0.65 | 0.71 |

This MAP is commonly found in real-world applications. We see that the performance suffers on all the platform except HD6970.

### 5.3.3 MAP-109

We see bandwidth improvement only on HD6970. On GTX280, the kernels with and without local memory perform similarly. There exists data reuse for threads with the same $ty$, but these memory requests are satisfied in one transaction (like broadcast).

Table 8: Performance impacts of using local memory for MAP-109

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 4.34 | 14.13 | 28.00 | 41.10 | 47.31 | 46.25 |
| w/i | 4.19 | 13.32 | 27.30 | 41.56 | 46.97 | 45.88 |
| sp | 0.97 | 0.94 | 0.98 | 1.01 | 0.99 | 0.99 |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 1.15 | 4.04 | 10.86 | 25.22 | 61.16 | 82.55 |
| w/i | 1.30 | 4.69 | 14.45 | 44.61 | 66.03 | 82.67 |
| sp | 1.13 | 1.16 | 1.33 | 1.77 | 1.08 | 1.00 |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 6.58 | 24.19 | 72.02 | 146.57 | 222.44 | 268.28 |
| w/i | 6.52 | 23.28 | 66.32 | 125.20 | 177.10 | 203.43 |
| sp | 0.99 | 0.96 | 0.92 | 0.85 | 0.80 | 0.76 |
| **E5620** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 2.57 | 7.42 | 16.18 | 24.16 | 29.46 | 18.85 |
| w/i | 2.40 | 5.47 | 8.51 | 10.56 | 11.22 | 9.32 |
| sp | 0.93 | 0.74 | 0.53 | 0.44 | 0.38 | 0.49 |

### 5.3.4 MAP-110

From Table 9, we see the memory bandwidth is significantly increased on GTX280. On HD6970 and GTX580, we also obtain some performance improvement. Furthermore, we suffer some bandwidth decrease on E5620. Using local memory for this MAP changes the access order of global memory, and thus we can ensure the coalesced memory access on GPUs.

Table 9: Performance impacts of using local memory for MAP-110

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 3.68 | 6.18 | 3.37 | 3.36 | 2.82 | 1.44 |
| w/i | 4.00 | 12.06 | 21.69 | 27.01 | 26.61 | 20.73 |
| sp | 1.09 | 1.95 | 6.43 | 8.03 | 9.44 | 14.41 |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 1.10 | 2.80 | 9.46 | 13.39 | 10.82 | 10.07 |
| w/i | 1.33 | 3.90 | 13.71 | 18.70 | 10.96 | 10.31 |
| sp | 1.21 | 1.40 | 1.45 | 1.40 | 1.01 | 1.02 |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 6.38 | 22.10 | 50.61 | 79.55 | 96.64 | 93.17 |
| w/i | 6.47 | 22.98 | 56.07 | 94.22 | 115.25 | 121.55 |
| sp | 1.01 | 1.04 | 1.11 | 1.18 | 1.19 | 1.30 |
| **E5620** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 2.68 | 6.45 | 12.63 | 16.23 | 13.38 | 11.63 |
| w/i | 2.46 | 5.66 | 9.17 | 11.37 | 8.95 | 8.28 |
| sp | 0.92 | 0.88 | 0.73 | 0.70 | 0.67 | 0.71 |

### 5.3.5 MAP-112

Table 10 shows significant performance increase on GTX280 and slight performance gain on GTX580. Using local memory on CPU E5620 suffers in memory bandwidth, while the performance varies on HD6970. Using local memory for this MAP changes the memory access orders of global references.

Table 10: Performance impacts of using local memory for MAP-112

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 3.79 | 6.11 | 3.32 | 3.15 | 2.46 | 1.38 |
| w/i | 3.85 | 10.83 | 15.72 | 15.88 | 14.38 | 11.00 |
| sp | 1.02 | 1.77 | 4.74 | 5.04 | 5.85 | 7.96 |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 1.11 | 2.78 | 7.49 | 11.08 | 6.38 | 5.70 |
| w/i | 1.13 | 4.52 | 10.83 | 11.69 | 6.40 | 5.29 |
| sp | 1.02 | 1.63 | 1.45 | 1.05 | 1.00 | 0.93 |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 6.32 | 21.41 | 48.26 | 72.61 | 86.78 | 78.67 |
| w/i | 6.37 | 21.65 | 49.96 | 78.11 | 93.89 | 93.59 |
| sp | 1.01 | 1.01 | 1.04 | 1.08 | 1.08 | 1.19 |
| **E5620** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 2.40 | 6.32 | 11.64 | 15.63 | 11.34 | 9.63 |
| w/i | 1.89 | 3.56 | 4.65 | 5.43 | 4.99 | 4.90 |
| sp | 0.79 | 0.56 | 0.40 | 0.35 | 0.44 | 0.51 |

### 5.3.6 MAP-113

Table 11: Performance impacts of using local memory for MAP-113

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 3.83 | 9.62 | 15.95 | 21.02 | 22.77 | 11.24 |
| w/i | 3.96 | 11.64 | 23.22 | 31.66 | 34.69 | 35.52 |
| sp | 1.03 | 1.21 | 1.46 | 1.51 | 1.52 | 3.16 |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 1.09 | 3.55 | 10.94 | 24.93 | 59.73 | 79.76 |
| w/i | 1.14 | 4.11 | 15.01 | 29.55 | 44.42 | 52.07 |
| sp | 1.04 | 1.16 | 1.37 | 1.19 | 0.74 | 0.65 |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 6.27 | 21.62 | 50.18 | 75.55 | 86.70 | 81.50 |
| w/i | 6.38 | 22.03 | 54.70 | 91.42 | 105.68 | 106.19 |
| sp | 1.02 | 1.02 | 1.09 | 1.21 | 1.22 | 1.30 |
| **E5620** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 2.69 | 6.20 | 6.90 | 15.49 | 11.24 | 9.52 |
| w/i | 1.93 | 3.75 | 4.24 | 5.81 | 5.00 | 4.68 |
| sp | 0.72 | 0.60 | 0.62 | 0.37 | 0.44 | 0.49 |

Table 11 shows that we get some performance improvement on GTX280 and GTX580, while suffer some performance loss on E5620 when using local memory. The memory bandwidth on HD6970 varies (first increases and then decreases) for different data sets. Using local memory for this MAP change the global reference order.

### 5.3.7 MAP-114

Table 12 sees the performance loss on GTX280, GTX580, and E5620, while it shows the memory bandwidth improves somewhat on HD6970. We have not found any factors incurring the performance increase.

Table 12: Performance impacts of using local memory for MAP-114

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 4.25 | 12.59 | 36.06 | 60.78 | 73.40 | 77.42 |
| w/i | 4.02 | 11.88 | 25.94 | 36.46 | 40.66 | 41.54 |
| sp | 0.95 | 0.94 | 0.72 | 0.60 | 0.55 | 0.54 |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 1.10 | 3.08 | 11.14 | 25.39 | 61.06 | 82.81 |
| w/i | 1.16 | 4.00 | 15.43 | 29.62 | 44.20 | 52.12 |
| sp | 1.06 | 1.30 | 1.39 | 1.17 | 0.72 | 0.63 |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 6.51 | 23.50 | 60.19 | 106.20 | 133.00 | 141.82 |
| w/i | 6.31 | 22.12 | 53.25 | 88.58 | 109.28 | 112.06 |
| sp | 0.97 | 0.94 | 0.88 | 0.83 | 0.82 | 0.79 |
| **E5620** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 2.49 | 6.77 | 13.78 | 21.01 | 15.80 | 11.33 |
| w/i | 1.98 | 3.82 | 5.18 | 5.95 | 5.66 | 5.48 |
| sp | 0.80 | 0.56 | 0.38 | 0.28 | 0.36 | 0.48 |

### 5.3.8 MAP-115

We see that the memory bandwidth is increased by using local memory on all the GPU platforms, while it suffers on CPU E5620. Using local memory for this MAP changes the access order of global memory, thus enabling the performance increase.

Table 13: Performance impacts of using local memory for MAP-115

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 3.55 | 7.20 | 15.59 | 19.75 | 12.50 | 5.91 |
| w/i | 3.97 | 11.69 | 24.70 | 35.26 | 39.26 | 39.95 |
| sp | 1.12 | 1.62 | 1.58 | 1.79 | 3.14 | 6.76 |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 1.08 | 2.90 | 10.24 | 24.52 | 59.87 | 77.09 |
| w/i | 1.12 | 4.38 | 14.54 | 29.14 | 43.42 | 51.09 |
| sp | 1.03 | 1.51 | 1.42 | 1.19 | 0.73 | 0.66 |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 6.12 | 21.30 | 45.57 | 75.38 | 84.64 | 79.47 |
| w/i | 6.29 | 22.35 | 54.40 | 92.15 | 106.76 | 106.93 |
| sp | 1.03 | 1.05 | 1.19 | 1.22 | 1.26 | 1.35 |
| **E5620** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 2.69 | 6.27 | 11.26 | 15.80 | 14.00 | 10.77 |
| w/i | 1.98 | 3.82 | 5.18 | 5.96 | 5.62 | 5.44 |
| sp | 0.73 | 0.61 | 0.46 | 0.38 | 0.40 | 0.50 |

### 5.3.9 MAP-116

Table 14 shows the bandwidth is increased on both GTX280 and GTX580, while it is not state on HD6970. On E5620, the performance suffers once more. With local memory, we are able to reuse data elements within one work-group and change the global reference order.

Table 14: Performance impacts of using local memory for MAP-116

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 3.37 | 6.97 | 16.03 | 19.60 | 12.79 | 6.25 |
| w/i | 4.06 | 10.86 | 25.50 | 35.41 | 39.25 | 40.24 |
| sp | 1.21 | 1.56 | 1.59 | 1.81 | 3.07 | 6.44 |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 1.12 | 3.60 | 11.17 | 28.28 | 64.31 | 82.37 |
| w/i | 1.24 | 4.44 | 14.27 | 39.67 | 54.58 | 66.07 |
| sp | 1.11 | 1.23 | 1.28 | 1.40 | 0.85 | 0.80 |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 6.30 | 21.64 | 53.12 | 86.39 | 91.51 | 89.02 |
| w/i | 6.28 | 22.24 | 58.19 | 100.52 | 119.42 | 120.33 |
| sp | 1.00 | 1.03 | 1.10 | 1.16 | 1.30 | 1.35 |
| **E5620** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 2.75 | 6.39 | 8.91 | 16.33 | 17.84 | 13.60 |
| w/i | 2.13 | 4.22 | 5.91 | 6.86 | 7.09 | 6.37 |
| sp | 0.77 | 0.66 | 0.66 | 0.42 | 0.40 | 0.47 |

### 5.3.10 MAP-204

Table 15: Performance impacts of using local memory for MAP-204

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 6.99 | 4.08 | 2.36 | 5.89 | | |
| w/i | 35.19 | 51.95 | 54.71 | 56.16 | | |
| bcr | 114.68 | 263.13 | 329.82 | 368.80 | | |
| sp1 | 5.03 | 12.73 | 23.14 | 9.54 | | |
| sp2 | 16.41 | 64.45 | 139.48 | 62.63 | | |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 22.09 | 51.66 | 70.18 | 79.83 | 82.19 | 81.85 |
| w/i | 60.77 | 135.94 | 234.41 | 261.21 | 266.52 | 236.56 |
| bcr | 68.06 | 163.69 | 288.30 | 328.28 | 336.44 | 272.22 |
| sp1 | 2.75 | 2.63 | 3.34 | 3.27 | 3.24 | 2.89 |
| sp2 | 3.08 | 3.17 | 4.11 | 4.11 | 4.09 | 3.33 |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 80.88 | 87.84 | 70.69 | 49.50 | 45.10 | |
| w/i | 130.26 | 180.00 | 189.43 | 190.89 | 191.03 | |
| bcr | 269.58 | 699.96 | 922.42 | 987.05 | 1007.11 | |
| sp1 | 1.61 | 2.05 | 2.68 | 3.86 | 4.24 | |
| sp2 | 3.33 | 7.97 | 13.05 | 19.94 | 22.33 | |
| **E5620** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 29.66 | 34.06 | 35.87 | 36.39 | 36.12 | |
| w/i | 28.50 | 34.43 | 37.36 | 38.05 | 38.36 | |
| bcr | 12.18 | 13.21 | 13.73 | 13.67 | 13.66 | |
| sp1 | 0.96 | 1.01 | 1.04 | 1.05 | 1.06 | |
| sp2 | 0.41 | 0.39 | 0.38 | 0.38 | 0.38 | |

The blank cells represent the execution time for this data set takes too long time ((for all?)). We significantly increase the memory bandwidth by using local memory on the GPU platform, while the performance is similar on E5620. Local memory is used to reuse data for work-items having the same *ty* index within one work-group. Furthermore, when reading data elements from local memory, all the requests falls into the same bank and different words, thus leading to bank conflicts. We remove the bank conflicts by using a shift value. Table 15 shows we dramatically improve the memory bandwidth further, especially NVIDIA GPUs. However, the optimization of bank-conflict removal incurs performance decrease on E5620.

### 5.3.11 MAP-205

Table 16: Performance impacts of using local memory for MAP-205

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 80.68 | 60.01 | 58.93 | 100.37 | 154.84 | 189.06 |
| w/i | 166.68 | 401.23 | 473.52 | 591.51 | 617.00 | 623.33 |
| sp | 2.07 | 6.69 | 8.03 | 5.89 | 3.98 | 3.30 |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 28.72 | 73.63 | 162.53 | 205.24 | 211.73 | 210.84 |
| w/i | 73.32 | 231.57 | 462.09 | 568.35 | 591.09 | 593.28 |
| sp | 2.55 | 3.15 | 2.84 | 2.77 | 2.79 | 2.81 |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 244.34 | 458.78 | 553.02 | 543.10 | 594.99 | 568.55 |
| w/i | 293.66 | 790.38 | 1023.29 | 1075.89 | 1083.91 | 1080.03 |
| sp | 1.20 | 1.72 | 1.85 | 1.98 | 1.82 | 1.90 |
| **E5620** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 63.29 | 90.05 | 99.04 | 101.60 | 102.04 | 102.10 |
| w/i | 72.46 | 123.27 | 143.49 | 152.57 | 151.04 | 145.79 |
| sp | 1.14 | 1.37 | 1.45 | 1.50 | 1.48 | 1.43 |

Table 16 shows we get better memory bandwidth from using local memory on all these four platforms. This improvement comes from the data reuse for work-items with the same $ty$ index within a work-group.

### 5.3.12 MAP-211

Table 17: Performance impacts of using local memory for MAP-211

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 7.80 | 3.97 | 2.31 | 5.58 | | |
| w/i | 34.63 | 50.97 | 53.79 | 55.34 | | |
| sp | 4.44 | 12.83 | 23.28 | 9.92 | | |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 21.78 | 42.81 | 46.47 | 60.10 | 70.35 | 70.94 |
| w/i | 58.22 | 143.15 | 221.18 | 244.56 | 247.91 | 213.72 |
| sp | 2.67 | 3.34 | 4.76 | 4.07 | 3.52 | 3.01 |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 73.37 | 82.94 | 76.06 | 45.63 | 44.08 | 38.28 |
| w/i | 126.48 | 173.96 | 182.92 | 184.01 | 184.50 | 184.07 |
| sp | 1.72 | 2.10 | 2.40 | 4.03 | 4.19 | 4.81 |
| **E5620** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 29.51 | 33.95 | 35.89 | 36.35 | 36.48 | |
| w/i | 27.06 | 32.93 | 35.69 | 36.44 | 35.96 | |
| sp | 0.92 | 0.97 | 0.99 | 1.00 | 0.99 | |

Table 17 shows we gain on memory bandwidth on the GPU platforms, and get similar performance on E5620. Local memory is used to reuse data elements (for work-items accessing the row indexed by $tx+ty$) and also change the way of accessing global memory.

### 5.3.13 MAP-302

From Table 18, we see dramatic bandwidth increase on GTX280, HD6970, GTX580. This increase comes from the data reuse and the changes in memory access orders. Without local memory, the performance deceases with the increase of data sets on E5620, while using local memory enables a stable memory bandwidth (**??**).

Table 18: Performance impacts of using local memory for MAP-302

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 85.03 | 101.31 | 124.66 | 130.62 | 131.30 | 131.67 |
| w/i | 150.75 | 352.14 | 446.74 | 469.40 | 476.37 | 479.55 |
| sp | 1.77 | 3.48 | 3.58 | 3.59 | 3.63 | 3.64 |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 25.61 | 82.24 | 131.16 | 147.33 | 148.89 | 148.75 |
| w/i | 73.39 | 231.66 | 455.72 | 558.51 | 580.41 | 583.12 |
| sp | 2.87 | 2.82 | 3.47 | 3.79 | 3.90 | 3.92 |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 219.53 | 476.66 | 531.04 | 467.28 | 404.78 | 397.83 |
| w/i | 283.98 | 752.59 | 967.97 | 1015.61 | 1027.50 | 1025.99 |
| sp | 1.29 | 1.58 | 1.82 | 2.17 | 2.54 | 2.58 |
| **E5620** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 13.75 | 13.87 | 8.47 | 5.73 | 1.54 | |
| w/i | 3.54 | 3.88 | 3.98 | 4.01 | 4.13 | |
| sp | 0.26 | 0.28 | 0.47 | 0.70 | 2.69 | |

### 5.3.14 MAP-303

Table 19: Performance impacts of using local memory for MAP-303

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 95.13 | 91.39 | 32.52 | 29.18 | 24.39 | |
| w/i | 150.94 | 317.01 | 299.07 | 302.18 | 280.16 | |
| sp | 1.59 | 3.47 | 9.20 | 10.36 | 11.49 | |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 25.47 | 76.84 | 134.64 | 150.88 | 151.28 | 151.38 |
| w/i | 73.76 | 236.42 | 455.72 | 556.26 | 525.51 | **170.32** |
| sp | 2.90 | 3.08 | 3.38 | 3.69 | 3.47 | 1.13 |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 223.94 | 495.02 | 604.88 | 582.88 | 509.87 | 338.67 |
| w/i | 281.23 | 751.67 | 967.20 | 1013.51 | 1024.08 | 1025.34 |
| sp | 1.26 | 1.52 | 1.60 | 1.74 | 2.01 | 3.03 |
| **E5620** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 13.96 | 18.02 | 15.39 | 5.85 | 2.67 | |
| w/i | 3.52 | 3.94 | 3.99 | 4.06 | 4.11 | |
| sp | 0.25 | 0.22 | 0.26 | 0.69 | 1.54 | |

The work-items with the same *ty* index within a work-group share the same column of data elements. Further, we change the access order of global memory access. Therefore, we see significant performance increase on GPUs in Table 19. The memory bandwidth on E5620 still suffers.

### 5.3.15 MAP-306

Table 20: Performance impacts of using local memory for MAP-306

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 43.17 | 38.72 | 76.57 | 77.81 | 77.64 | |
| w/i | 136.49 | 292.21 | 367.02 | 384.91 | 390.05 | |
| sp | 3.16 | 7.55 | 4.79 | 4.95 | 5.02 | |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 27.77 | 79.37 | 131.51 | 151.19 | 152.22 | 151.49 |
| w/i | 72.79 | 224.71 | 422.74 | 505.22 | 522.16 | 508.16 |
| sp | 2.62 | 2.83 | 3.21 | 3.34 | 3.43 | 3.35 |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 202.88 | 377.56 | 360.54 | 459.06 | 456.87 | 317.90 |
| w/i | 256.50 | 605.30 | 754.13 | 756.06 | 770.09 | 772.45 |
| sp | 1.26 | 1.60 | 2.09 | 1.65 | 1.69 | 2.43 |
| **E5620** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 4.94 | 7.23 | 3.18 | 7.90 | 1.31 | 1.13 |
| w/i | 2.63 | 3.88 | 3.96 | 4.19 | 4.20 | 4.29 |
| sp | 0.53 | 0.54 | 1.25 | 0.53 | 3.22 | 3.80 |

Table 20 shows we achieve significant performance improvement on the GPU platforms. This is due to data reuse for work-items with the same $(tx+ty)$. On E5620, the bandwidth varies on different data sets.

### 5.3.16 MAP-401

We see (from Table 21) that we gain on memory bandwidth on all the used platforms. This improvement results from the data reuse.

Table 21: Performance impacts of using local memory for MAP-401

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 102.24 | 132.79 | 148.15 | | | |
| w/i | 331.56 | 445.74 | 460.58 | | | |
| sp | 3.24 | 3.36 | 3.11 | | | |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 127.76 | 161.97 | 162.96 | | | |
| w/i | 737.11 | 990.76 | 1032.69 | | | |
| sp | 5.77 | 6.12 | 6.34 | | | |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 618.19 | 637.16 | 646.21 | 615.93 | | |
| w/i | 1219.92 | 1285.82 | 1299.19 | 1302.21 | | |
| sp | 1.97 | 2.02 | 2.01 | 2.11 | | |
| **E5620** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 95.99 | 96.75 | 100.04 | | | |
| w/i | 144.18 | 143.26 | 147.48 | | | |
| sp | 1.50 | 1.48 | 1.47 | | | |

### 5.3.17 MAP-407

Table 22 shows we significantly increase the memory bandwidth on NVIDIA GPUs (GTX280 and GTX580). We also see some performance increase on HD6970 and E5620. The performance increase comes from the data sharing for work-items with the same $tx$. Furthermore, when using local memory of moderate size, we see further bandwidth increase on GTX280, no change on HD6970 and GTX580, and performance decrease on E5620. Therefore, the M-approach does not always perform better than the L-approach.

Table 22: Performance impacts of using local memory for MAP-407

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 11.85 | 13.20 | 12.07 | 12.33 | 12.41 | 5.70 |
| w/i L | 40.19 | 69.47 | 81.87 | 86.09 | 87.30 | 87.34 |
| w/i M | 57.09 | 113.31 | 145.24 | 158.77 | 161.41 | 162.21 |
| sp L | 3.39 | 5.26 | 6.78 | 6.98 | 7.04 | 15.33 |
| sp M | 4.82 | 8.59 | 12.04 | 12.88 | 13.01 | 28.47 |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 9.15 | 27.54 | 52.90 | 75.48 | 84.93 | 85.16 |
| w/i L | 26.55 | 60.77 | 90.74 | 130.48 | 145.31 | 148.54 |
| w/i M | 26.13 | 55.10 | 90.82 | 128.04 | 142.93 | 146.23 |
| sp L | 2.90 | 2.21 | 1.72 | 1.73 | 1.71 | 1.74 |
| sp M | 2.86 | 2.00 | 1.72 | 1.70 | 1.68 | 1.72 |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 54.33 | 70.71 | 79.69 | 70.76 | 79.52 | 70.17 |
| w/i L | 106.37 | 197.74 | 273.97 | 301.02 | 308.20 | 310.56 |
| w/i M | 103.06 | 182.61 | 249.30 | 272.69 | 279.86 | 281.59 |
| sp L | 1.96 | 2.80 | 3.44 | 4.25 | 3.88 | 4.43 |
| sp M | 1.90 | 2.58 | 3.13 | 3.85 | 3.52 | 4.01 |
| **E5620** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 8.30 | 8.79 | 9.39 | 9.59 | 9.66 | 9.68 |
| w/i L | 12.60 | 14.79 | 16.02 | 16.29 | 16.42 | 16.52 |
| w/i M | 8.63 | 9.37 | 9.92 | 10.09 | 10.12 | 10.13 |
| sp L | 1.52 | 1.68 | 1.71 | 1.70 | 1.70 | 1.71 |
| sp M | 1.04 | 1.07 | 1.06 | 1.05 | 1.05 | 1.05 |

### 5.3.18 MAP-408

This MAP is commonly found in real-world applications like Image Convolution. We see (from Table 23) that with local memory the memory bandwidth is increased on GTX280, HD6970, GTX580, and E5620. The performance increase comes from reuse of overlapped data.

Table 23: Performance impacts of using local memory for MAP-408

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 35.98 | 56.40 | 67.26 | 72.28 | 73.72 | 73.71 |
| w/i | 37.90 | 64.59 | 74.65 | 79.10 | 79.92 | 80.19 |
| sp | 1.05 | 1.15 | 1.11 | 1.09 | 1.08 | 1.09 |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 13.40 | 28.64 | 61.04 | 87.99 | 98.75 | 100.41 |
| w/i | 19.29 | 56.14 | 87.97 | 126.06 | 140.51 | 143.74 |
| sp | 1.44 | 1.96 | 1.44 | 1.43 | 1.42 | 1.43 |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 76.73 | 139.88 | 176.41 | 154.29 | 179.63 | 189.31 |
| w/i | 98.17 | 196.86 | 258.92 | 282.92 | 290.41 | 292.46 |
| sp | 1.28 | 1.41 | 1.47 | 1.83 | 1.62 | 1.54 |
| **E5620 (5)** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 10.45 | 11.59 | 11.99 | 12.14 | 12.18 | 12.19 |
| w/i | 11.77 | 14.11 | 14.91 | 15.35 | 15.33 | 15.35 |
| sp | 1.13 | 1.22 | 1.24 | 1.26 | 1.26 | 1.26 |

### 5.3.19 MAP-409

Table 24: Performance impacts of using local memory for MAP-409

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 36.10 | 58.32 | 35.01 | 34.49 | 32.47 | 29.06 |
| w/i L | 40.22 | 69.31 | 79.54 | 85.06 | 86.12 | 86.49 |
| w/i M | 57.31 | 111.92 | 134.19 | 150.01 | 149.64 | 148.04 |
| sp L | 1.11 | 1.19 | 2.27 | 2.47 | 2.65 | 2.98 |
| sp M | 1.59 | 1.92 | 3.83 | 4.35 | 4.61 | 5.09 |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 13.53 | 31.83 | 64.20 | 89.88 | 102.58 | 104.69 |
| w/i L | 25.26 | 62.29 | 90.54 | 130.49 | 146.33 | 150.01 |
| w/i M | 26.15 | 61.91 | 91.92 | 130.32 | 145.93 | 149.71 |
| sp L | 1.87 | 1.96 | 1.41 | 1.45 | 1.43 | 1.43 |
| sp M | 1.93 | 1.94 | 1.43 | 1.45 | 1.42 | 1.43 |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 84.95 | 135.57 | 168.81 | 180.23 | 183.79 | 184.88 |
| w/i L | 105.18 | 189.26 | 263.69 | 293.17 | 300.94 | 303.16 |
| w/i M | 103.86 | 187.71 | 259.00 | 287.43 | 295.72 | 297.98 |
| sp L | 1.24 | 1.40 | 1.56 | 1.63 | 1.64 | 1.64 |
| sp M | 1.22 | 1.38 | 1.53 | 1.59 | 1.61 | 1.61 |
| **E5620** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 28.56 | 42.03 | 48.25 | 51.19 | 52.01 | 52.01 |
| w/i L | 18.38 | 23.58 | 26.09 | 27.05 | 27.31 | 27.28 |
| w/i M | 18.69 | 23.76 | 26.26 | 27.10 | 27.31 | 27.31 |
| sp L | 0.64 | 0.56 | 0.54 | 0.53 | 0.53 | 0.52 |
| sp M | 0.65 | 0.57 | 0.54 | 0.53 | 0.53 | 0.53 |

Table 24 shows we can achieve performance improvement on GTX280, HD6970, and GTX580, while the bandwidth suffers on E5620. The performance increase results from data reuse among work-items with the same $ty$ index.

### 5.3.20 MAP-410

Table 25: Performance impacts of using local memory for MAP-410

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 12.01 | 6.74 | 1.85 | 1.77 | 1.26 | 29.08 |
| w/i | 34.94 | 52.84 | 64.03 | 64.02 | 65.11 | 149.03 |
| sp | 2.91 | 7.84 | 34.63 | 36.13 | 51.57 | 5.13 |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 13.21 | 28.88 | 49.64 | 45.98 | 33.12 | 16.01 |
| w/i | 22.97 | 61.42 | 86.11 | 121.62 | 74.40 | 67.79 |
| sp | 1.74 | 2.13 | 1.73 | 2.64 | 2.25 | 4.24 |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 44.64 | 62.04 | 63.55 | 64.41 | 68.72 | 56.36 |
| w/i | 94.64 | 162.04 | 208.99 | 229.34 | 233.91 | 235.05 |
| sp | 2.12 | 2.61 | 3.29 | 3.56 | 3.40 | 4.17 |
| **E5620** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 8.14 | 9.90 | 10.50 | 10.65 | 10.52 | 10.40 |
| w/i | 5.82 | 7.05 | 7.44 | 7.59 | 7.78 | 7.70 |
| sp | 0.72 | 0.71 | 0.71 | 0.71 | 0.74 | 0.74 |

Table 25 sees significant bandwidth increase on GPU platforms, while the bandwidth decreases on E5620 using local memory. Data reuse and transposed memory access patterns are the two factors incurring performance increase.

### 5.3.21 MAP-412

Table 26: Performance impacts of using local memory for MAP-412

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 12.79 | 6.62 | 1.87 | 1.75 | 1.28 | |
| w/i | 36.42 | 61.48 | 60.41 | 58.37 | 55.63 | |
| sp | 2.85 | 9.28 | 32.29 | 33.37 | 43.34 | |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 12.85 | 26.98 | 26.81 | 17.02 | 9.44 | 5.88 |
| w/i | 21.83 | 61.82 | 88.37 | 123.44 | 55.95 | 35.00 |
| sp | 1.70 | 2.29 | 3.30 | 7.25 | 5.92 | 5.95 |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 45.36 | 50.54 | 50.35 | 50.65 | 52.72 | 36.73 |
| w/i | 102.30 | 192.60 | 258.07 | 282.93 | 291.25 | 293.42 |
| sp | 2.26 | 3.81 | 5.13 | 5.59 | 5.52 | 7.99 |
| **E5620** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 2.93 | 6.67 | 10.46 | 10.60 | 10.38 | 10.21 |
| w/i | 2.75 | 5.29 | 7.08 | 7.01 | 7.18 | 7.04 |
| sp | 0.94 | 0.79 | 0.68 | 0.66 | 0.69 | 0.69 |

Using local memory, we see (from Table 26) that bandwidth improves dramatically on GTX280, HD6970, and GTX580 due to data reuse and changes on memory access patterns. Meanwhile, using local memory on E5620 gets decreased bandwidth.

### 5.3.22 MAP-413

Table 27: Performance impacts of using local memory for MAP-413

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 12.76 | 13.45 | 11.91 | 12.33 | 12.38 | 5.96 |
| w/i | 38.66 | 66.11 | 76.83 | 80.42 | 81.61 | 81.78 |
| sp | 3.03 | 4.91 | 6.45 | 6.52 | 6.59 | 13.73 |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 13.73 | 28.58 | 53.99 | 73.72 | 80.31 | 80.26 |
| w/i | 21.61 | 61.60 | 88.47 | 125.51 | 139.39 | 141.84 |
| sp | 1.57 | 2.16 | 1.64 | 1.70 | 1.74 | 1.77 |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 32.05 | 53.94 | 52.27 | 46.74 | 51.83 | 47.40 |
| w/i | 99.70 | 193.80 | 256.59 | 279.34 | 287.66 | 289.86 |
| sp | 3.11 | 3.59 | 4.91 | 5.98 | 5.55 | 6.12 |
| **E5620** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 3.27 | 7.03 | 10.43 | 10.62 | 10.55 | 10.38 |
| w/i | 2.98 | 5.31 | 6.78 | 7.10 | 6.99 | 7.21 |
| sp | 0.91 | 0.76 | 0.65 | 0.67 | 0.66 | 0.69 |

Table 27 shows that using local memory on GPU platforms much better than the kernel without it. The performance increase comes from data sharing and changing of global memory access. In spite of this, the memory bandwidth suffers on E5620.

### 5.3.23 MAP-414

Table 28: Performance impacts of using local memory for MAP-414

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 33.87 | 34.45 | 68.21 | 72.04 | 72.60 | 72.11 |
| w/i | 39.24 | 67.41 | 79.34 | 83.05 | 84.39 | 84.68 |
| sp | 1.16 | 1.96 | 1.16 | 1.15 | 1.16 | 1.17 |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 13.41 | 30.45 | 62.56 | 88.05 | 97.85 | 99.20 |
| w/i | 26.05 | 61.77 | 89.07 | 127.24 | 141.97 | 145.49 |
| sp | 1.94 | 2.03 | 1.42 | 1.45 | 1.45 | 1.47 |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 73.92 | 128.37 | 146.24 | 175.88 | 190.06 | 181.24 |
| w/i | 101.85 | 192.24 | 261.12 | 287.68 | 295.09 | 297.44 |
| sp | 1.38 | 1.50 | 1.79 | 1.64 | 1.55 | 1.64 |
| **E5620** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 3.40 | 6.32 | 11.48 | 11.77 | 11.76 | 11.77 |
| w/i | 2.96 | 4.69 | 7.07 | 6.95 | 6.66 | 7.00 |
| sp | 0.87 | 0.74 | 0.62 | 0.59 | 0.57 | 0.59 |

For this MAP, data reuse is possibly the only source of performance improvement. The memory bandwidth is slightly improved on GPUs platforms, while it still suffers on E5620 (in Table 28).

### 5.3.24 MAP-415

Table 29 shows the memory bandwidth is significantly improved GTX280, HD6970, and GTX580. The performance improvement mainly comes from data reuse and changing of global memory access. Meanwhile, on E5650 using local memory suffers in performance.

Table 29: Performance impacts of using local memory for MAP-415

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 9.07 | 6.72 | 13.00 | 12.52 | 5.67 | 3.01 |
| w/i | 39.28 | 68.22 | 79.40 | 83.07 | 84.37 | 84.36 |
| sp | 4.33 | 10.15 | 6.11 | 6.63 | 14.87 | 27.98 |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 12.96 | 27.00 | 53.75 | 75.08 | 82.94 | 81.49 |
| w/i | 20.45 | 61.72 | 88.87 | 126.45 | 140.72 | 143.96 |
| sp | 1.58 | 2.29 | 1.65 | 1.68 | 1.70 | 1.77 |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 36.90 | 55.27 | 52.18 | 64.36 | 70.07 | 54.92 |
| w/i | 101.75 | 192.03 | 261.50 | 287.92 | 291.47 | 297.47 |
| sp | 2.76 | 3.47 | 5.01 | 4.47 | 4.16 | 5.42 |
| **E5620** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 4.58 | 4.35 | 6.27 | 11.78 | 11.76 | 11.72 |
| w/i | 3.87 | 3.81 | 5.07 | 6.91 | 6.74 | 7.02 |
| sp | 0.85 | 0.88 | 0.81 | 0.59 | 0.57 | 0.60 |

### 5.3.25 MAP-416

Table 30 shows the memory bandwidth is significantly improved GTX280, HD6970, and GTX580. The performance improvement mainly comes from data reuse and changing of global memory access. Meanwhile, on E5650 using local memory suffers in performance.

Table 30: Performance impacts of using local memory for MAP-416

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 8.72 | 6.83 | 12.99 | 12.52 | 6.13 | 3.24 |
| w/i | 37.58 | 65.27 | 76.87 | 80.79 | 82.25 | 80.71 |
| sp | 4.31 | 9.56 | 5.92 | 6.45 | 13.41 | 24.92 |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 12.54 | 26.98 | 52.99 | 76.10 | 83.47 | 84.00 |
| w/i | 20.89 | 61.76 | 88.53 | 126.14 | 140.37 | 143.64 |
| sp | 1.67 | 2.29 | 1.67 | 1.66 | 1.68 | 1.71 |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 37.37 | 55.23 | 51.96 | 63.21 | 48.42 | 49.58 |
| w/i | 102.57 | 193.30 | 256.99 | 287.59 | 294.16 | 295.87 |
| sp | 2.74 | 3.50 | 4.95 | 4.55 | 6.08 | 5.97 |
| **E5620** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 1.96 | 3.23 | 11.55 | 11.79 | 11.77 | 11.76 |
| w/i | 1.84 | 2.95 | 7.29 | 7.66 | 7.33 | 7.33 |
| sp | 0.94 | 0.91 | 0.63 | 0.65 | 0.62 | 0.62 |

### 5.3.26 MAP-507

Table 31: Performance impacts of using local memory for MAP-507

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 7.09 | 12.31 | 13.61 | 14.49 | 14.58 | 6.92 |
| w/i L | 11.75 | 33.34 | 64.25 | 85.37 | 92.32 | 93.79 |
| w/i M | 11.95 | 35.99 | 72.84 | 99.10 | 108.36 | 110.50 |
| sp1 | 1.66 | 2.71 | 4.72 | 5.89 | 6.33 | 13.55 |
| sp2 | 1.69 | 2.92 | 5.35 | 6.84 | 7.43 | 15.96 |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 3.24 | 10.77 | 25.75 | 51.99 | 92.31 | 108.34 |
| w/i L | 3.57 | 13.33 | 47.34 | 86.67 | 132.67 | 156.32 |
| w/i M | 3.70 | 13.96 | 52.13 | 93.69 | 146.77 | 174.97 |
| sp1 | 1.10 | 1.24 | 1.84 | 1.67 | 1.44 | 1.44 |
| sp2 | 1.14 | 1.30 | 2.02 | 1.80 | 1.59 | 1.62 |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 17.11 | 44.76 | 75.89 | 86.03 | 97.68 | 97.37 |
| w/i L | 19.31 | 65.00 | 159.84 | 253.58 | 295.83 | 301.75 |
| w/i M | 19.56 | 66.23 | 169.08 | 276.57 | 327.07 | 330.53 |
| sp1 | 1.13 | 1.45 | 2.11 | 2.95 | 3.03 | 3.10 |
| sp2 | 1.14 | 1.48 | 2.23 | 3.21 | 3.35 | 3.39 |
| **E5620** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 4.95 | 9.50 | 16.97 | 19.14 | 19.12 | 19.30 |
| w/i L | 1.06 | 1.23 | 1.28 | 1.40 | 1.36 | 1.37 |
| w/i M | 1.05 | 1.23 | 1.28 | 1.40 | 1.36 | 1.37 |
| sp1 | 0.21 | 0.13 | 0.08 | 0.07 | 0.07 | 0.07 |
| sp2 | 0.21 | 0.13 | 0.08 | 0.07 | 0.07 | 0.07 |

For this MAP, the performance increase (on GTX280, HD6970, and GTX580) mainly comes from data reuse (of work-items with the same $tx$ index) and changes in memory access orders. Note that using local memory

for this MAP perform much worse than the case without it. Furthermore, M-approach performs better on the GPU platforms.

### 5.3.27 MAP-508

Table 32: Performance impacts of using local memory for MAP-508

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 11.58 | 29.97 | 53.36 | 71.01 | 78.13 | 79.27 |
| w/i | 11.15 | 30.32 | 55.10 | 71.63 | 77.44 | 79.12 |
| sp | 0.96 | 1.01 | 1.03 | 1.01 | 0.99 | 1.00 |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 3.33 | 10.60 | 31.77 | 88.70 | 180.45 | 225.41 |
| w/i | 3.43 | 12.72 | 38.49 | 74.69 | 111.29 | 120.66 |
| sp | 1.03 | 1.20 | 1.21 | 0.84 | 0.62 | 0.54 |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 19.09 | 62.87 | 135.13 | 185.18 | 218.87 | 240.46 |
| w/i | 18.19 | 59.82 | 130.15 | 181.69 | 211.22 | 219.20 |
| sp | 0.95 | 0.95 | 0.96 | 0.98 | 0.97 | 0.91 |
| **E5620** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 5.53 | 11.36 | 16.90 | 20.06 | 16.58 | 16.57 |
| w/i | 0.90 | 1.05 | 1.06 | 1.09 | 1.08 | 1.06 |
| sp | 0.16 | 0.09 | 0.06 | 0.05 | 0.06 | 0.06 |

Using local memory brings few performance gains, due to little data reuse. Thus, on GTX280 and GTX580, we see a similar performance. Meanwhile, the speedup varies a lot on HD6970 for different data sets. Furthermore, using local memory performs much worse on E5620.

### 5.3.28 MAP-509

Table 33: Performance impacts of using local memory for MAP-509

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 12.39 | 36.43 | 46.39 | 46.65 | 44.34 | 40.85 |
| w/i L | 11.78 | 33.20 | 65.12 | 80.42 | 83.19 | 82.41 |
| w/i M | 12.02 | 35.27 | 72.42 | 92.62 | 94.22 | 96.25 |
| sp1 | 0.95 | 0.91 | 1.40 | 1.72 | 1.88 | 2.02 |
| sp2 | 0.97 | 0.97 | 1.56 | 1.99 | 2.12 | 2.36 |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 3.38 | 12.22 | 32.00 | 72.71 | 187.71 | 248.48 |
| w/i L | 3.72 | 13.33 | 34.86 | 88.28 | 143.86 | 156.20 |
| w/i M | 3.94 | 13.99 | 51.03 | 94.41 | 159.76 | 175.07 |
| sp1 | 1.10 | 1.09 | 1.09 | 1.21 | 0.77 | 0.63 |
| sp2 | 1.17 | 1.14 | 1.59 | 1.30 | 0.85 | 0.70 |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 19.55 | 67.59 | 173.60 | 304.45 | 409.08 | 466.16 |
| w/i L | 19.02 | 65.59 | 161.15 | 253.02 | 301.21 | 316.21 |
| w/i M | 19.30 | 67.31 | 168.13 | 274.50 | 332.32 | 350.60 |
| sp1 | 0.97 | 0.97 | 0.93 | 0.83 | 0.74 | 0.68 |
| sp2 | 0.99 | 1.00 | 0.97 | 0.90 | 0.81 | 0.75 |
| **E5620** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 4.66 | 10.77 | 14.34 | 19.19 | 19.15 | 19.32 |
| w/i L | 1.07 | 1.26 | 1.28 | 1.35 | 1.34 | 1.35 |
| w/i M | 1.08 | 1.26 | 1.28 | 1.35 | 1.34 | 1.36 |
| sp1 | 0.23 | 0.12 | 0.09 | 0.07 | 0.07 | 0.07 |
| sp2 | 0.23 | 0.12 | 0.09 | 0.07 | 0.07 | 0.07 |

The memory bandwidth of using local memory varies on different data sets and platforms, i.e., no consistent

increase or decrease in memory bandwidth. Furthermore, M-approach performs slightly better than the L-approach.

### 5.3.29 MAP-510

Table 34: Performance impacts of using local memory for MAP-510

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 7.28 | 12.35 | 12.68 | 13.23 | 13.08 | 6.99 |
| w/i | 11.02 | 29.09 | 53.41 | 66.60 | 67.21 | 62.59 |
| sp | 1.51 | 2.36 | 4.21 | 5.03 | 5.14 | 8.95 |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 3.22 | 8.97 | 24.37 | 50.81 | 88.51 | 102.76 |
| w/i | 3.43 | 12.36 | 43.49 | 73.77 | 110.68 | 127.10 |
| sp | 1.07 | 1.38 | 1.78 | 1.45 | 1.25 | 1.24 |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 16.60 | 41.23 | 67.65 | 75.69 | 84.93 | 85.89 |
| w/i | 18.26 | 59.02 | 131.02 | 188.53 | 215.90 | 221.63 |
| sp | 1.10 | 1.43 | 1.94 | 2.49 | 2.54 | 2.58 |
| **E5620** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 4.60 | 11.54 | 16.30 | 18.87 | 17.18 | 17.20 |
| w/i | 0.91 | 1.02 | 1.09 | 1.06 | 1.09 | 1.07 |
| sp | 0.20 | 0.09 | 0.07 | 0.06 | 0.06 | 0.06 |

Table 34 shows that the memory bandwidth is improved significantly on GPU platforms, while using local memory suffers a lot on E5620. The performance improvement mainly comes from transposed memory accesses.

### 5.3.30 MAP-512

Table 35 shows the bandwidth is increased from using local memory on GPUs. This is due to the changes on memory access orders.

Table 35: Performance impacts of using local memory for MAP-512

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 7.48 | 12.33 | 13.38 | 13.44 | 13.36 | 7.38 |
| w/i | 10.92 | 27.97 | 49.21 | 62.06 | 66.48 | 65.55 |
| sp | 1.46 | 2.27 | 3.68 | 4.62 | 4.98 | 8.89 |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 3.17 | 9.06 | 21.39 | 51.19 | 90.37 | 102.70 |
| w/i | 3.33 | 12.78 | 34.00 | 54.69 | 86.38 | 95.90 |
| sp | 1.05 | 1.41 | 1.59 | 1.07 | 0.96 | 0.93 |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 16.42 | 37.18 | 64.50 | 66.49 | 82.32 | 84.88 |
| w/i | 17.83 | 53.41 | 110.80 | 151.34 | 169.52 | 173.13 |
| sp | 1.09 | 1.44 | 1.72 | 2.28 | 2.06 | 2.04 |
| **E5620** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 4.83 | 8.80 | 15.50 | 17.80 | 15.45 | 15.02 |
| w/i | 0.90 | 1.02 | 1.04 | 1.09 | 1.06 | 1.10 |
| sp | 0.19 | 0.12 | 0.07 | 0.06 | 0.07 | 0.07 |

### 5.3.31 MAP-513

We see that the memory bandwidth is improved on NVIDIA GPUs significantly, while it varies for different data sets on HD6970. Changes on memory access orders are the main sources of the performance improvement. Using local memory on E5620 still performs much worse than the cases without it.

Table 36: Performance impacts of using local memory for MAP-513

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 7.54 | 12.66 | 13.95 | 14.57 | 14.52 | 7.33 |
| w/i | 10.83 | 28.47 | 50.73 | 62.86 | 66.93 | 67.47 |
| sp | 1.44 | 2.25 | 3.64 | 4.31 | 4.61 | 9.21 |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 3.23 | 8.90 | 21.28 | 49.19 | 95.32 | 109.60 |
| w/i | 3.42 | 11.66 | 32.94 | 55.49 | 88.04 | 97.97 |
| sp | 1.06 | 1.31 | 1.55 | 1.13 | 0.92 | 0.89 |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 16.21 | 38.29 | 64.23 | 64.54 | 82.90 | 84.91 |
| w/i | 17.98 | 53.34 | 113.63 | 157.51 | 177.86 | 178.60 |
| sp | 1.11 | 1.39 | 1.77 | 2.44 | 2.15 | 2.10 |
| **E5620** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 3.13 | 11.50 | 11.02 | 17.94 | 15.09 | 14.57 |
| w/i | 0.89 | 1.01 | 1.05 | 1.07 | 1.06 | 1.09 |
| sp | 0.28 | 0.09 | 0.10 | 0.06 | 0.07 | 0.07 |

### 5.3.32 MAP-514

We achieve similar performance on NVIDIA GPUs for cases without and with local memory. The performance varies on HD6970, and suffers a lot on E5620. There are very few sharing neighbor data elements.

Table 37: Performance impacts of using local memory for MAP-514

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 11.29 | 26.15 | 54.52 | 71.92 | 77.69 | 77.03 |
| w/i | 11.34 | 31.80 | 61.26 | 79.64 | 84.40 | 86.95 |
| sp | 1.00 | 1.22 | 1.12 | 1.11 | 1.09 | 1.13 |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 2.69 | 9.34 | 25.28 | 83.01 | 178.51 | 193.65 |
| w/i | 3.47 | 13.19 | 35.39 | 63.15 | 102.64 | 115.29 |
| sp | 1.29 | 1.41 | 1.40 | 0.76 | 0.57 | 0.60 |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 18.89 | 61.56 | 126.41 | 187.67 | 215.68 | 216.13 |
| w/i | 17.99 | 59.15 | 126.25 | 188.93 | 212.32 | 219.53 |
| sp | 0.95 | 0.96 | 1.00 | 1.01 | 0.98 | 1.02 |
| **E5620** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 3.60 | 11.01 | 15.78 | 18.78 | 16.12 | 15.82 |
| w/i | 0.90 | 0.93 | 1.05 | 1.06 | 1.09 | 1.08 |
| sp | 0.25 | 0.08 | 0.07 | 0.06 | 0.07 | 0.07 |

### 5.3.33 MAP-515

MAP-515 changes the reference orders of global memory, and thus we get significant performance improvement on NVIDIA GPUs. The memory bandwidth varies on HD6970, and again using local memory incurs bad performance on E5620.

Table 38: Performance impacts of using local memory for MAP-515

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 7.08 | 9.91 | 13.74 | 14.26 | 7.13 | 3.71 |
| w/i | 11.09 | 30.86 | 60.14 | 77.77 | 82.92 | 80.92 |
| sp | 1.57 | 3.11 | 4.38 | 5.45 | 11.64 | 21.79 |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 9.46 | 22.65 | 49.46 | 94.76 | 109.20 | 193.65 |
| w/i | 13.12 | 35.65 | 61.27 | 98.73 | 111.25 | 115.29 |
| sp | 1.39 | 1.57 | 1.24 | 1.04 | 1.02 | 0.60 |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 15.82 | 40.29 | 57.08 | 79.26 | 83.20 | 79.90 |
| w/i | 18.10 | 59.20 | 128.77 | 192.80 | 215.21 | 218.55 |
| sp | 1.14 | 1.47 | 2.26 | 2.43 | 2.59 | 2.74 |
| **E5620** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 3.69 | 12.09 | 16.10 | 18.78 | 17.03 | 17.98 |
| w/i | 0.92 | 1.06 | 1.10 | 1.12 | 1.06 | 1.07 |
| sp | 0.25 | 0.09 | 0.07 | 0.06 | 0.06 | 0.06 |

### 5.3.34 MAP-516

Table 39 shows using local memory enables a memory bandwidth increase on GPUs, but decrease on E5620. Furthermore, the M-approach performs better than the L-approach.

Table 39: Performance impacts of using local memory for MAP-516

| **GTX280** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
|---|---|---|---|---|---|---|
| w/o | 7.20 | 10.45 | 14.16 | 14.57 | 7.80 | 3.97 |
| w/i L | 11.17 | 30.32 | 59.02 | 74.80 | 79.30 | 73.99 |
| w/i M | 11.55 | 30.73 | 63.87 | 83.55 | 89.69 | 86.46 |
| sp1 | 1.55 | 2.90 | 4.17 | 5.14 | 10.17 | 18.62 |
| sp2 | 1.60 | 2.94 | 4.51 | 5.74 | 11.50 | 21.76 |
| **HD6970** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 3.25 | 10.04 | 26.61 | 57.67 | 97.51 | |
| w/i L | 3.68 | 13.11 | 39.75 | 82.32 | 124.00 | |
| w/i M | 3.94 | 13.72 | 48.99 | 85.40 | 132.58 | |
| sp1 | 1.13 | 1.31 | 1.49 | 1.43 | 1.27 | |
| sp2 | 1.21 | 1.37 | 1.84 | 1.48 | 1.36 | |
| **GTX580** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 14.84 | 41.06 | 56.48 | 87.63 | 92.87 | 88.58 |
| w/i L | 18.27 | 61.56 | 143.93 | 220.06 | 249.41 | 254.95 |
| w/i M | 18.27 | 62.39 | 149.51 | 235.97 | 274.13 | 281.57 |
| sp1 | 1.23 | 1.50 | 2.55 | 2.51 | 2.69 | 2.88 |
| sp2 | 1.23 | 1.52 | 2.65 | 2.69 | 2.95 | 3.18 |
| **E5620** | 128x128 | 256x256 | 512x512 | 1024x1024 | 2048x2048 | 4096x4096 |
| w/o | 5.21 | 10.95 | 15.78 | 18.60 | 18.51 | 17.91 |
| w/i L | 0.98 | 1.10 | 1.20 | 1.22 | 1.25 | 1.20 |
| w/i M | 0.98 | 1.11 | 1.21 | 1.23 | 1.25 | 1.21 |
| sp1 | 0.19 | 0.10 | 0.08 | 0.07 | 0.07 | 0.07 |
| sp2 | 0.19 | 0.10 | 0.08 | 0.07 | 0.07 | 0.07 |

### 5.4 Summary

To summarize, we get the following observations and summaries:

- **On GPUs (GTX280, HD6970, GTX580):** using local memory benefits the memory bandwidth, where the improvement comes from not only data reuse but changes on memory access orders.
- **GTX280 vs. GTX580:** using local memory obtains more in performance on GTX280 than on GTX580 due to the introduction of caches on GTX580.
- **On CPU E5620:** we get better performance for MAP-205, 401, and 408 by using local memory.
- **M-approach vs. L-approach:** the M-approach does not always perform better than the L-approach.

## 6 Case Study: Enabling Local Memory

Based on our performance database, we propose a hybrid approach for automated usage of local memory, shown in Figure 6. Given a kernel, we abstract MAP for each memory reference. Depending on the given platform and the MAP, we search the performance record in the performance database. If it is beneficial, we will perform code transformation to enable local memory on the input kernel. Otherwise, we keep the original kernel code.

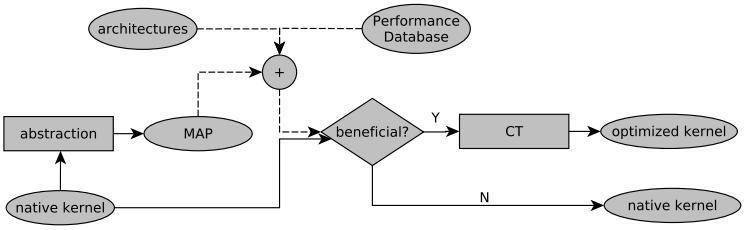



Figure 8: A hybrid framework of enabling local memory.

## 7 Conclusions

Architecture diversity and their implementation differences makes the performance benefits of using local memory unpredictable only via modeling. Thus, in this paper, we present a benchmark-based approach to tackle this issue. We first present a two-part approach to describe memory access patterns for many-thread applications. For each MAP, we design a set of benchmarks of native versions (without local memory) and optimized versions (using local memory). Then we evaluate the microbenchmarks on NVIDIA GPUs (GTX280, GTX580), AMD GPUs (HD6970), and Intel CPUs (E5620), and get a performance database. This database can provide essential information for further work of enabling local memory.

# References

[1] Muthu M. Baskaran, Uday Bondhugula, Sriram Krishnamoorthy, J. Ramanujam, Atanas Rountev, and P. Sadayappan. Automatic data movement and computation mapping for multi-level parallel architectures with explicitly managed memories. In *Proceedings of PPoPP*, 2008. 6

[2] M. Kandemir and A. Choudhary. Compiler-directed scratch pad memory hierarchy design and management. In *Proceedings of DAC*. ACM, 2002. 6

[3] Jianbin Fang, Ana L. Varbanescu, Jie Shen, and Henk Sips. ELMO: A User-Friendly API to enable local memory in OpenCL kernels. In *Proceedings of the 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP'13)*, 2013. 6

[4] Armin Grösslinger. Precise management of scratchpad memories for localising array accesses in scientific codes. In *Proceedings of the 18th International Conference on Compiler Construction: Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009*, volume 5501 of *CC '09*, pages 236–250, Berlin, Heidelberg, 2009. Springer-Verlag. 6

[5] Intel Inc. *Intel OpenCL Optimization Guide*, April 2012. 6, 8

[6] Byunghyun Jang, D. Schaa, P. Mistry, and D. Kaeli. Exploiting memory access patterns to improve memory performance in Data-Parallel architectures. *IEEE Transactions on Parallel and Distributed Systems*, 22(1):105–118, January 2011. 8