Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/070651

Methods for Optimizing OpenCL Applications on Heterogeneous Multicore Architectures

Slo-Li Chu[*] *and Chih-Chieh Hsiao*

Department of Information and Computer Engineering, Chung Yuan Christian University, Chung Li, 32023, Taiwan

Received: 28 Feb. 2013, Revised: 26 Jun. 2013, Accepted: 27 Jun. 2013
Published online: 1 Nov. 2013

Abstract: Heterogeneous multicore architectures with CPU and add-on GPUs or streaming processors are now widely used in computer systems. These GPUs provide substantially more computation capability and memory bandwidth compared to traditional multi-cores. Also, because they are highly programmable, they provide the computational performance needed for realistic graphics rendering. Applications with general computations can also be leveraged onto these GPUs. This study discusses the architectures of these highly efficient GPUs and applies a unified programming standard called OpenCL to fully utilize their capabilities. Despite their great potential, applications of these GPUs are challenging because of their diverse underlying architectural characteristics. In this study, several optimizing techniques are applied on OpenCL-compatible heterogeneous multicore architectures to achieve thread-level and data-level parallelisms. The architectural implications of these techniques are discussed. Finally, optimization principles for these architectures will be are proposed. The experimental reveal average speedups of 24 and 430 for non-optimized and optimized kernels, respectively.

Keywords: Parallel computing, GPU computing, OpenCL, Optimizing techniques for OpenCL, GPU.

1 Introduction

Traditional microprocessor designs are reaching performance limits due to power wall from increased frequency and circuit area and memory wall from the performance gap between CPU and memory. Diminishing returns on instruction-level parallelism have also increased the difficulty of scaling performance as predicted by Moore Law. Recently, the programmability of these add-on graphics processing units (GPU) and streaming processors has increased due to demands for increased realism in 3D games and graphics applications. Therefore, general-purpose computing on these devices is now possible. Accordingly, almost every computer system now has a heterogeneous platform with CPU and GPU or streaming processor to provide both graphics rendering and general-purpose computations. Today, GPUs provide substantially more computational power compared to state-of-the-art CPUs, and the performance gap between them is expected to increase over time. In 2004, the fastest Pentium4 achieved 7 GFLOPs whereas the ATi Radeon X850 achieved 66 GFLOPs. The Cell processor in Sony PlayStation3 introduced in 2005 achieved over 200 GFLOPs. The fastest six-core Intel Core i7-980X currently achieves over 100 GFLOPs whereas the fastest GPU ATi Radeon HD5870 achieves 2720 GFLOPs. Thus, exploiting the ever increasing computing power of modern GPUs is a challenge. In the past, writing parallel programs for these high-performance heterogeneous computer systems required familiarity with graphics APIs or vendor-specific APIs. These APIs and programming paradigms are extremely difficult to implement [26–30]. The most popular parallel programming paradigms, such as OpenMP [15] and MPI, are unsuitable for these heterogeneous multicore architectures. Vendor-specific GPGPU APIs, such as Sh [1], Brook [2], CAL [22] are hard to programming and porting across varied architectures. Therefore OpenCL [4] is proposed to easily program and migrate between diverse architectures. Although OpenCL is computationally powerful and compatible with different platforms, fully utilizing OpenCL devices requires careful tuning of computing kernels. Figure 1 compares speedups between optimized and non-optimized kernels. The OpenCL version benchmark ”Calculation of Pi” uses integral to approachn value with computationally expensive operations. The

[*] Corresponding author e-mail: slchu@cycu.edu.tw

baseline of the speedup is the sequential version that runs on Intel Core i5-750. The speedups significantly differ between optimized and non-optimized kernels, which provide sufficient optimization space for performance tuning on OpenCL-compatible devices.

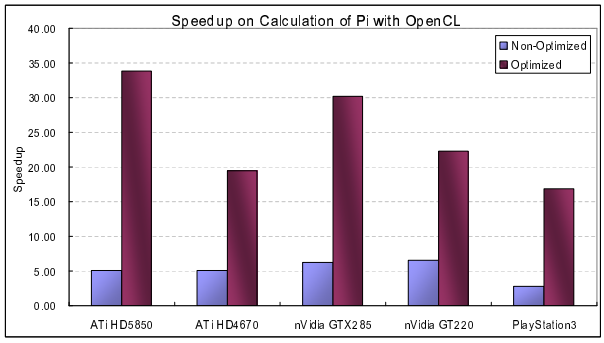


Fig. 1: The speedup difference between optimized and non-optimized OpenCL "Pi calculation" kernels..

Research in OpenCL has focused on algorithm porting [5, 6] or programming paradigm comparisons [7]. The limited research in kernel code optimization has resulted in the underutilization of these devices. Similar programming paradigms such as CUDA have been studied for many years to enable their use on these computing devices [8, 9, 16]. However, optimization techniques on CUDA are specified on heterogeneous multicore architectures with nVidia GPUs, which may be inappropriate for various OpenCL-compatible devices. This study presents several workloads with OpenCL and discusses the architectural implications of the underlying hardware. Several optimization techniques such as vectorization, tiling memory access, and data redistribution are then discussed. The proper combinations of the above optimizing techniques are discussed. Their performance differences are also examined. Finally, suggestions for enhancing performance are given. This paper is organized as follows. Section 2 presents the OpenCL parallel programming paradigm and gives a simple example. Section 3 discusses the architectural details of OpenCL compatible computing devices. Section 4 describes optimization techniques for overcoming architectural limitations. The experimental results for speedup comparisons between OpenCL-compatible devices and suggested optimization methods are given in section 5. Finally, conclusions are given in section 6.

2 OpenCL Programming Paradigms

OpenCL is a standard for parallel programming for modern heterogeneous multicore architectures. The purpose of OpenCL is to provide a compatible code for different devices, architectures, and applications. Therefore, CPUs, GPUs, and other accelerators can be used to accelerate computation-intensive or data parallel applications.

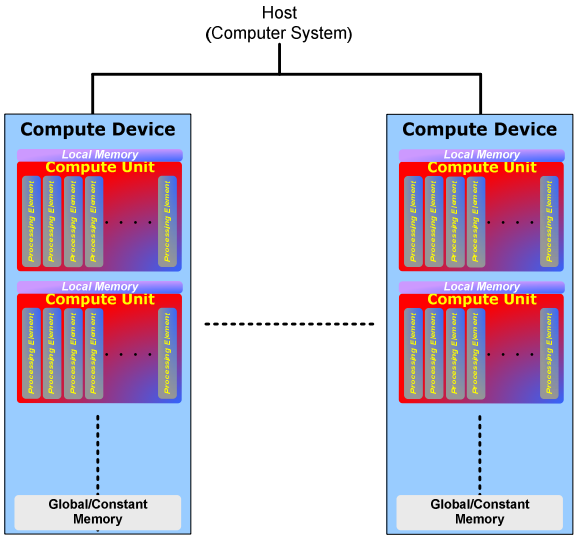


Fig. 2: The OpenCL platform model.

Figure 2 shows the OpenCL platform model. The host is the processor of the modern computer system that provides the executing platform of OpenCL runtime for dispatch workloads for OpenCL computing devices. The computing device may be a CPU, GPU, DSP or an accelerator such as SPE in Cell, which executes OpenCL kernels. Each device has several computing units consisting of multiple processing elements and its device memory to store input data. Each computing device consists of multiple processing elements. Once an OpenCL kernel is launched, the host must specify executing dimensions (three maximum) and an N-dimensional range. The kernel is executed in multiple instances called work-items within this range. Multiple work-items can also be combined into a work group to enable communication and synchronization within a group. The number of work-items per work group is specified by a host program written by programmers. The work-items within a work group are executed concurrently on the processing elements of a single compute unit.

3 Architecture Overview and Execution Model of OpenCL Computing Devices

This section describes the hardware platforms and the mappings between OpenCL and these devices.

3.1 ATi Radeon HD5800 series

The Radeon HD5800 series [10], announced by ATi in 2009, has over 2 teraflops computing capability. Figure 3 shows the HD5800 architecture. This architecture combines SIMD and VLIW engines. It consists of 20 SIMD engines; each SIMD engine is as wide as 16 SIMD machines. Each element, i.e., thread processor, has a five-issue, VLIW design consisting of four basic stream cores. These floating-point operations are IEEE754 compatible with limited rounding mode support. Then it contains one engine for special functions called T-Unit, mainly responsible for sine, cosine, and reciprocal functions. A branch unit in each thread processor solves simple branch problems during execution. The mapping from OpenCL computation kernel on ATi Radeon HD5800 series is also illustrated in Fig. 3. The OpenCL kernel is composed by work groups, each consisting of multiple wavefronts. The wavefront, which is the basic hardware execution unit of ATi GPU, executes N number of work-items concurrently. The Radeon HD4670 consists of only 8 SIMD engines with 8 thread processors each. An SIMD engine maintains multiple wavefronts and switches between them to hide execution latency, but only one wavefront can be executed at one time. To finish an instruction of all threads in a wavefront, the thread processors must use the same instructions to operate on different data from different threads in four consecutive cycles.

3.2 nVidia GeForce GT200 series

In early 2009, nVidia introduced its GT200 series [11] GPU, which has over 1 teraFLOPs computing capability. The GT200 architecture, as shown in Fig. 4, consists of 10 TPCs (Texture/Processor Clusters), each of which contains three SMs (Streaming Multiprocessors) and one Texture unit. The main computing elements in GT200 are the 8 SPs (Streaming Processors), which operate in SIMD (Single Instruction Multiple Data) fashion. Both SPs and SFUs (Special Function Units) share the same instruction, data and instruction L1 cache, instruction fetch and dispatch unit in a SM. A GT200 GPU contains 240 SPs. The mapping from OpenCL kernel to nVidia GT200 series is shown in Fig. 4. Each OpenCL kernel is a composite of multiple workgroups, and each can be mapped as a thread block in GT200. A thread block consists of multiple warps, which contain 32 work-items. The Global Block Scheduler schedules thread blocks onto each TPC; each SM in a TPC is scheduled to work on a warp in block. Since each warp consists of 32 computing threads, the 8 SPs require 4 consecutive cycles for lock-step execution of one instruction of all 32 threads in a warp. To fully exploit processor power, each SM usually keeps multiple warps in flight and switches between them to hide latencies. High thread-level parallelism is thus avoided. The warp or workgroup switches after an instruction of all threads in a warp is completed.

3.3 IBM Cell Processor

The architecture of IBM Cell processor [12] is illustrated in Fig. 5. PPE (Power Processor Element) is the main control unit of Cell processor, and SPEs (Synergistic Processor Elements) are specifically designed for data-intensive and streaming processing computations. The PPE is a PowerPC processor which is an in-order, 2-way simultaneous multi-threading, 64-bit Power architecture with VMX extension, with L2 cache. SPE (Synergistic Processing Element) is a dual-issue 128-bit SIMD architecture with deep pipelining. All of these processor elements, memory controller, and I/O are attached onto EIB (Element Interconnect Bus), which is presently implemented as a circular ring comprising four 16B-wide unidirectional channels with counter-rotating in pairs. To access external memory, the SPEs only can rely on DMA to move data from/to memory controller. Meanwhile, the memory controller is shared by all processor elements. So it limits the memory bounded applications and programming paradigm. Additionally, the Cell processor adopted in Sony PlayStation3 only enables six SPEs for programmers.

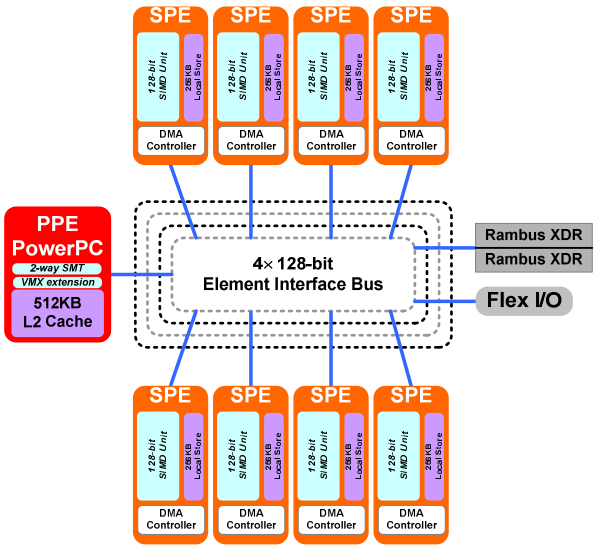


Fig. 5: The organization of IBM Cell processor.

3.4 Parameters of Evaluated OpenCL Devices

Table 1 summarizes the architectures considered in this study. Some of the entries are inconsistent with previous

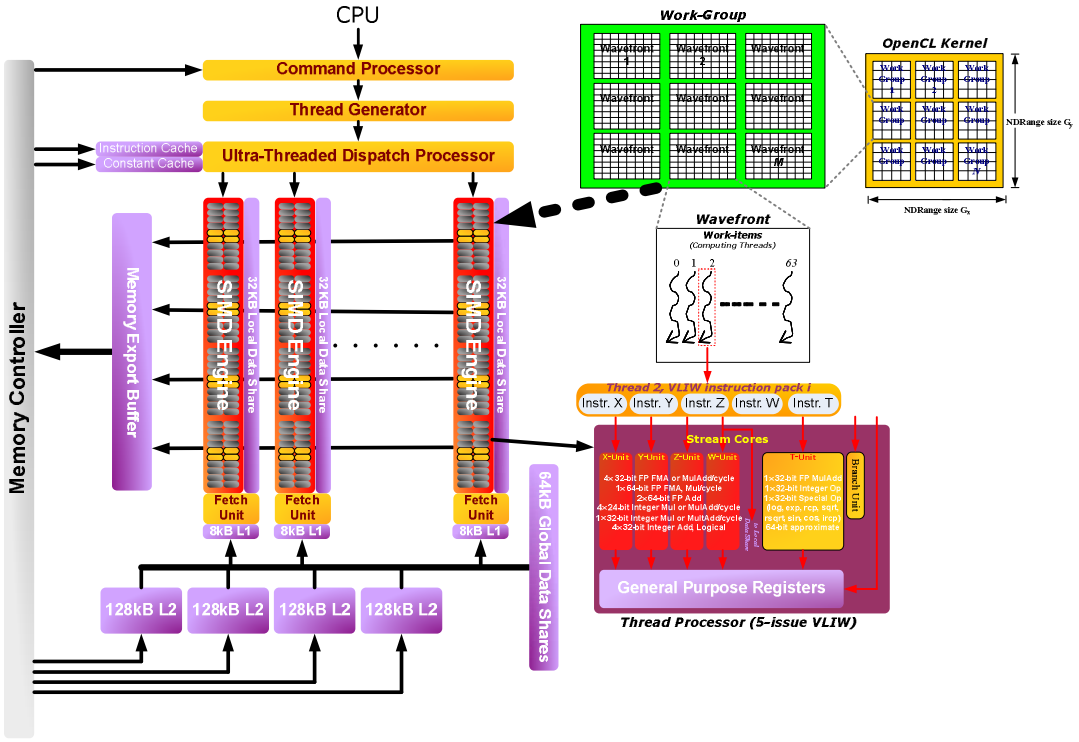


Fig. 3: The architecture of ATi Radeon HD5800 series with OpenCL mapping.

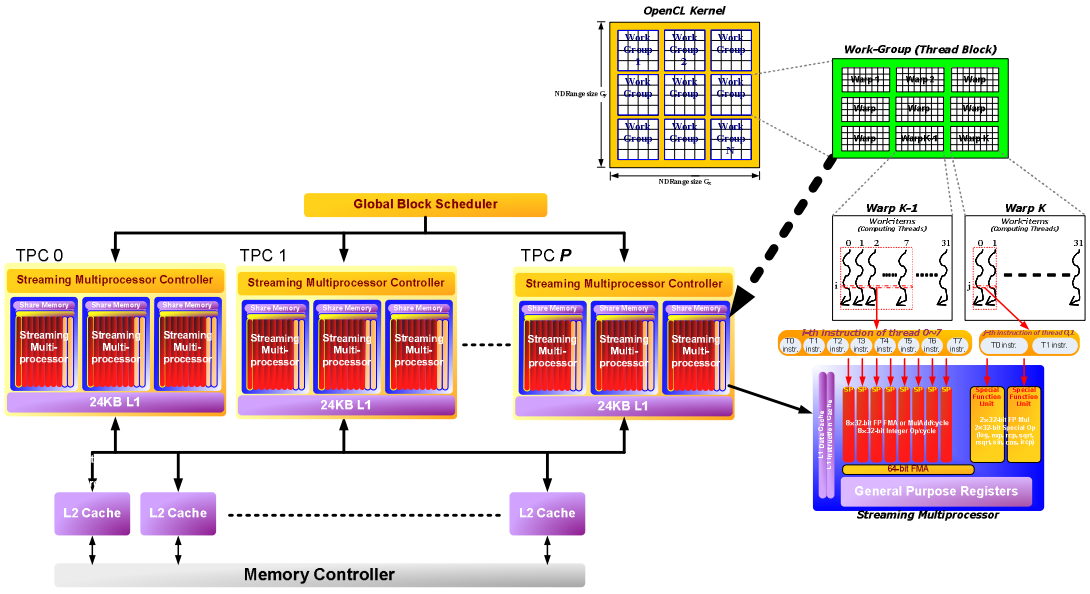


Fig. 4: The architecture of nVidia GT200 series with OpenCL mapping.

discussions since high-end versions were not tested in this study (e.g., HD5870 has 1,600 PEs whereas the HD5850 analyzed in this study has only has 1,440 PEs, and GT220 has only two TPCs consisting of 6 SMs). Except for the thread processors in their SIMD engines and their wavefront size, ATi Radeon HD4000 and HD5000 have similar computing architectures

Table 1: The summary of architectures used in this study.

	IBM Cell in PlayStation3	nVidia GeForce GT220	nVidia GeForce GTX285	ATi Radeon HD4670	ATi Radeon HD5850
Clock rate of PE	3.2GHz	1.36GHz	1.476GHz	750MHz	725MHz
Number of PE	1+6	48	240	320	1440
Number of SIMD Engines	6	6	30	8	18
GFLOPS (single)	204	196	1062.72	480	2088
GFLOPS (double)	15	N/A	89	N/A	418
Warp/Wavefront size	N/A	32	32	32	64
Memory interface	XDR	GDDR3	GDDR3	DDR3	GDDR5
Memory bandwidth	25.6GB/s	25.3GB/s	159.0GB/s	32.0GB/s	128.0GB/s
Transistor count	241M	486M	1.4B	514M	2.15B
Chip area	235mm^2	100mm^2	576mm^2	146mm^2	334mm^2
TDP	110W	58W	183W	59W	151W
Fabrication Process	90nm	40nm	55nm	55nm	40nm

4 Optimizing Methods for OpenCL Programs

This section describes two benchmarks that reveal the architectural implications of these OpenCL compatible platforms. One is computation-intensive application and one is memory-intensive program. Several optimization techniques, including massive multithreading, vectorization, tiling, and data redistribution, are also performed to determine their effects on performance. The first is ”Pi calculation”, which uses integrals to approximate the value of . Because of its high computational expense, inter-loop independents are easily dividable into varying numbers of work-items, which makes it suitable for massive multithreading and vectorization tests to enhance the performance of computing devices with thread-level, data-level, and instruction-level parallelisms. The compiled kernel also uses less than ten registers (five for ATi GPU and seven for nVidia GPU), which eliminate concerns about register pressure [25] and keeps the computing device busy with a huge amount of threads. The second benchmark is matrix multiplication with two 2048 X 2048 matrices. Since it requires large workloads in memory access operations, typical optimization techniques such as tiling and data redistribution can be applied to exploit its parallelism. Different combinations of optimization techniques are also evaluated using this benchmark. The performance enhancements discussed in this section are demonstrated in the speedup comparisons with its sequential version executed in OpenCL host. The OpenCL speedups on GPUs are based on its sequential C version running on the Intel Core i5-750 of the host whereas where the speedups on PlayStation3 are based on its sequential C version running on PowerPC Processing Element against Synergistic Processing Elements. To observe performance enhancements on computing devices, the execution times for OpenCL kernels described in this section do not include compilation and data transfer between the host and the computing device. The execution times of GPUs are measured by the vendor-provided profiler with microsecond accuracy. The performance breakdown given at the end of this section is measured by host OS timer. Table 2 shows the platform environments used in following sections. All performance test results given in the following sections are average values of multiple runs.

Table 2: The summary of experimental environments.

	IBM Cell in PlayStation3	nVidia GeForce GT220	nVidia GeForce GTX285	ATi Radeon HD4670	ATi Radeon HD5850
OS	Linux 2.6.21 (64-bit)	Windows7 (32-bit)			
Host	Power Processor Element	Intel Core-i5 750			
Compiler	GCC 4.1.2	Visual Studio 2008			
OpenCL runtime	Cell SDK 3.1.0 + OpenCL SDK 0.1.1	GPU Computing SDK 2.3A		StreamSDK 2.0.1	
Driver	N/A	Forceware 190.89		Catalyst 10.2	

4.1 Massive Multithreading for OpenCL Programs

The current GPU designs rely on switching between massive multithreading to hide execution latencies in ALU and memory accesses. The appropriate number of computing threads or work-items should be carefully set to fully utilize massive processing elements and hide latencies. The number of work-items in a workgroup was set to 16, 32 and 64. Also, the number of total work-items was varied to observe its effects on performance and to observe how architecture affects the number of work-items. The selected number of iterations was 1,000,857,600, which was devisable for all configurations in this study.

Figure 6 shows the speedups on ”Pi calculation” with various numbers of work-items. Since the size of the hardware wavefront in HD5850 was 64, 16 and 32 work-items per workgroup could only occupy one half and one quarter of the wavefront, respectively, which degraded performance. HD5850 needed at least 64 work-items per workgroup to form a full wavefront and to

achieve full hardware utilization. The performance enhancement reached optimal when the total number of work-items was set to 4,608. Accordingly, wavefront size is 64; the number of SIMD engines is 18. To fully utilize all thread processors in HD5850 with a computationally expensive program, each workgroup or its integer multiple must contain 64 work-items.

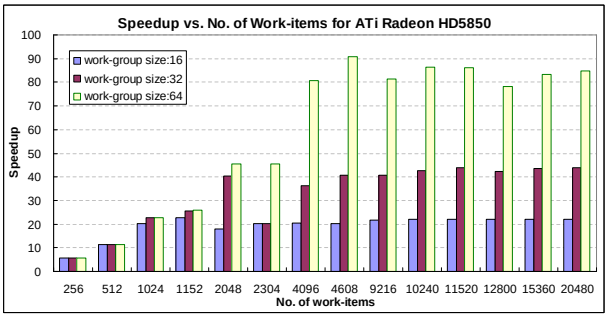


Fig. 6: Speedup with different number of total work-items and work-items per work-groups in ATi Radeon HD5850 versus its sequential version.

Figure 7 shows the speedup of different configurations with ”Pi calculation” on ATi Radeon HD4670. The size of the hardware wavefront in HD5850 is 32 instead of 64, and the test results show that the larger workgroup cannot enhance performance due to hardware limitations. The best performance is obtained by a workgroup size of 32 and 1,024 work-items. For a wavefront size of 32, the number of SIMD engines is 8. If the number of work-items per workgroup is less than 32, performance drops because the hardware wavefront is not fully occupied.

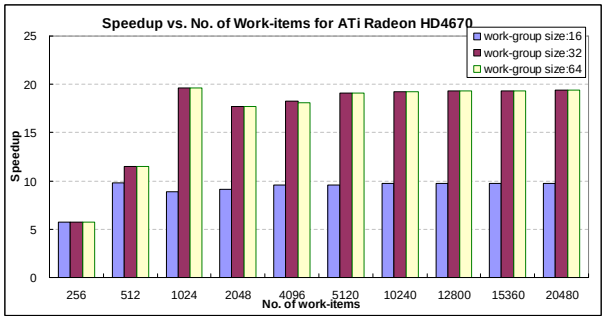


Fig. 7: Speedup with different number of total work-items and work-items per workgroups in ATi Radeon HD4670 versus its sequential version.

Figure 8 shows the speedups obtained with different parameters of OpenCL kernel in nVidia GeForce GTX285. The number of basic working units in hardware is a warp in nVidia GPU, which is 32 work-items. Experimental results indicate that doubling the number of work-items per workgroup obtains a better performance, which is 64. If this number is set higher than 128, performance can exceed 64. According to the profiling results, performance is best when the number of work-items or computing threads is larger than or equal to 512 per SM. Each workgroup turns into one thread block in nVidia GPU, and each SM can have maximum 8 thread blocks with 1,024 computing threads in flight according to its resource availability.

The best performance was obtained when each workgroup consisted of 64 work-items, The Occupancy is 0.5, which equals 512 per SM. The hardware limitation makes 1,024 computing threads within one SM. Also, if the compiled kernel has a large number of register requirements, Occupancy is reduced lower since the register pressure limits the number of in-flight computing threads in hardware.

The number of SIMD engine is 30 for GTX285, which is the minimum number to keep the GPU busy with 15,360 threads. That is, the nVidia GTX285 can accommodate tens of thousands more threads than ATi GPUs can. The minimum number of computing threads required to hide latencies per SM is 64 instead of 16 in ATi GPU since the clock rate of the nVidia PE is twice as fast as that of ATi GPU with deeper pipeline latency to hide.

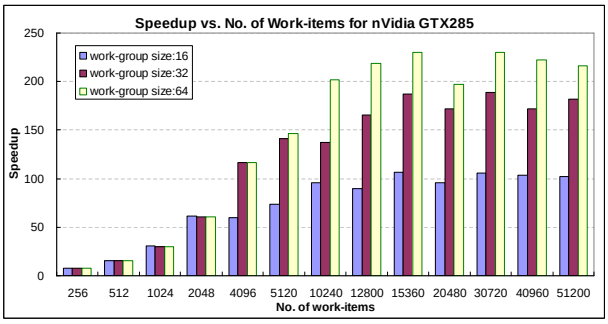


Fig. 8: Speedup with different number of total work-items and work-items per work-groups in nVidia GeForce GTX285 versus its sequential version.

Figure 9 shows the speedup of nVidia GeForce GT220. The best performance occurs when workgroup size is 64. An abnormal point occurs when total number of work-items equals 4,096, where a workgroup size of 16 performs better than a size of 32. The occupancy is 0.5 when workgroup size is 64, but each TPC is responsible for 28 workgroups when workgroup size is 32. To finish the computation, each SM must finish one third of workgroups in a TPC, which leaves one workgroup for either workgroup size 64 or 32. Accordingly, 64 and 32 threads left for processing on one SM cause this abnormal

point. The best performance for GT220 is shown when total number of workgroup is 3,072.

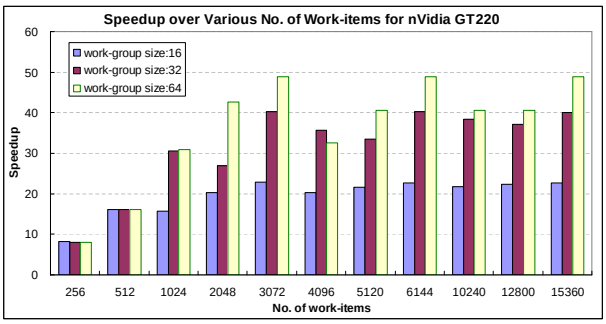


Fig. 9: Performance with different number of total work-items and work-items per work-groups in nVidia GeForce GT220 versus its sequential version.

Figure 10 presents the speedup on IBM Cell processor in PlayStation3 with ”Pi calculation”. Since the SPE is not hardware multithreaded, the increase in work-items per workgroup does not help performance, so the experiment here uses only one work-item per workgroup. The experimental results show that the best performance occurs when using all available SPEs in PlayStation3 and no more.

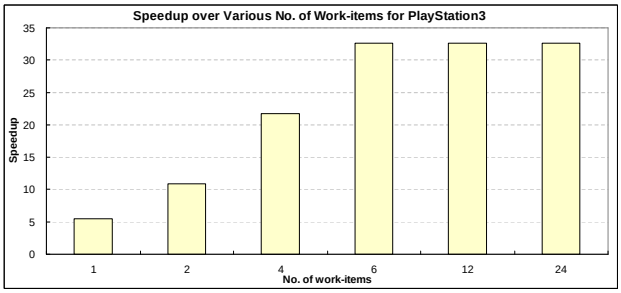


Fig. 10: Speedup with different number of total work-items in IBM Cell processor versus its sequential version.

4.2 Vectorization for OpenCL Programs

This subsection illustrates performance and architecture implications with vectorization. As the architectural discussion in section 3, some of the architectures are VLIW or SIMD design, which relies on compiler to rearrange the code into appropriate instruction slots or vector lanes. If the writing of kernel program can consider writing in vectorized form, the compiler can rearrange the kernel source to increase the efficiency of these devices. Again, the ”Pi calculation” is used as benchmark to illustrate the effect of vectorization. The total numbers of work-items are 4,608 and 1,024 for ATi HD5850 and HD4670, respectively, 30,720 and 6,144 for GTX285 and GT220, respectively, and 6 for PlayStation3. The size of workgroup is 32 for HD4670, 1 for PlayStation3 and 64 for others. The OpenCL provides vector primitive like float2, float4, float8, float16 and double2 to present vector type of floating-point. To write OpenCL kernel into vectorized version requires only minor modifications if it is vectorizable.

4.3 Tiling and Data Redistribution for OpenCL Programs

This subsection discusses optimization techniques with tiling, data redistribution and various combinations of optimization techniques. Since each computing device in an OpenCL platform model has its own memory, all computing units must access this memory to process resident data. Therefore, access to device memory should be carefully managed to avoid conflicts or redundancies. Conventional tiling and data redistribution mechanisms can reduce the pressure on the memory subsystems of OpenCL computing devices. The adopted benchmark is ”Matrix Multiplication”. The number of work-item is launched with two dimension range, which consists of (matrix width/4) x (matrix height/4) work-items in total. Each work-item is responsible for a 4 x 4 region of result matrix C. Workgroup size is 32 for ATi Radeon HD4670, 64 for all other GPU devices and 1 for PlayStation3. In a naive implementation, each element within its 4 x 4 region performs intuitive multiply and accumulates operations in all rows in matrix A and in all columns in matrix B. To discover the computing power from SIMD or VLIW designs in ATi GPUs or IBM Cell, the inner loop of naive kernel is unrolled and jammed with vector primitive.

4.4 The Combinations of the Optimizing Methods

Figure 11 shows the experimental results for various combinations of optimization techniques in terms of speedup, compared with Intel Core i5. The speedup is obtained from the ratio of execution time of OpenCL version program on various OpenCL devices over the execution time of sequential version program on host Intel Core i5 processor. Excluded the setup and compiling time of OpenCL program, the execution time here is only adopted the net execution time of OpenCL kernel. Although many-core OpenCL compatible computing devices must access their global memory without a well-designed pattern, their high bandwidths still enable performance enhancements. Since the nVidia GT220,

which is the low-end GPU in market, has insufficient hardware resources for executing multiplication tasks for two 2048x2048 matrices, a 1024x1024 matrix was run on nVidia GT220. The speedups in PlayStation3 were insignificant because memory access was limited by its EIB ring interconnection network to its memory subsystem. However, the speedup on the naive version and on all vectorized or data redistribution versions were about 2.2 against its sequential version on PPE due to this limitation. In ATi GPUs, vectorization of the naive kernel can expand memory bandwidth and enhance computational capability due to drawbacks in the VLIW design. However, the speedup improves until vector length reaches 16 and 8 for HD5850 and HD4670, respectively. These speedup reductions result from register pressure because the length of the native vector register of ATi GPUs is only four single-precision floating-point values. Because nVidia GPUs have a scalar design, the benefits of vectorization are minimal [23]. Tiling helps all architectures to reuse read in data and significantly improve performance. All temporal registers are declared with scalar variables when use tiling alone. Therefore, register pressure degrades HD5850 performance when tile size is increased to 4x4. The HD4670 still exhibits enhanced performance because each workgroup has only 32 work-items instead of 64, which reduces register pressure. For both nVidia GPUs, performance continues to scale up due to data reuse. Notably, the PlayStation3 is 8.5 times and 31.8 times faster than its previous version when tile size is 2x2 and 4x4, respectively. If the data distribution, processing elements read a portion of the element from one row of matrix A and then multiply and accumulate it while directly accessing matrix B. Since it is not vectorized or tiled, it does not benefit overall performance during execution and may even degrade performance if the barrier requires synchronization.

The following paragraph reports the results of performance tests of various combinations of the above optimization techniques. Data redistribution combined with vectorization enhances computational capacity only in ATi GPUs. In HD5850, this combination is 20% faster than vectorization alone. Use of data redistribution with tiling, which puts one input tile into local memory, can help to reduce register pressure from local variables generated by unroll-and-jam. Therefore, performance significantly improves. When vectorization is combined with tiling, the native vector register can be applied in ATi GPUs, which greatly reduces register pressure and significantly increases the number of in-flight threads and performance. Finally, all optimization techniques were combined. This final version of the kernel improved performance for some GPUs. The lack of improvement is due to driver issues on GT220 and local memory mapping issues on HD4670. The speedup in GT285 was lower than that in HD5850 because vectorization increased register pressure and reduced occupancy from 0.5 to 0.25. PlayStation3 performance was also limited by its effective bandwidth, which limited the maximum speedup obtained from this kernel is about 32.

In the final optimization version, the ATi Radon HD5850 performed nearly as twice as fast as the nVidia GTX285, which indicated that program optimization is more important for HD5850 than for nVidia GPUs. Also, the computations of this benchmark are multiply and add, which requires all five stream cores in thread processor. Accordingly, the theoretical performance of ATi HD5850 can be determined only if the kernel is carefully tuned and no special function is needed in T-unit.

4.5 Summary of Optimization of Pi Calculation

This subsection shows that overall speedup includes kernel compilation time and memory from/to OpenCL computing devices on selected configurations and then gives the performance breakdown. The configurations in this subsection are the optimal numbers of work-items and vectorization with float8 for all GPUs and float4 for PlayStation3. Due to the inefficiency of the PlayStation3 compiler, compilation is performed offline. Figure 12 shows the overall speedup.

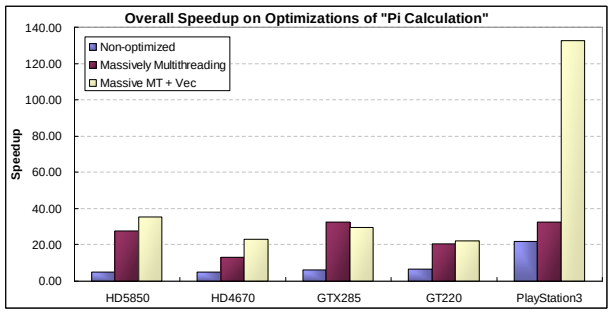


Fig. 12: Overall speedups on optimizations of ”Pi Calculation” versus its sequential version on host.

If only execution time is considered, speedups are significantly lower. Some are even slower when vectorization is applied. To identify the cause of the reduced speedup, Compile, Memory setup, Execution and Memory read back, were analyzed separately. Figure 13 shows the performance results.

The overall performance shows multithreading (MT) obtains performance improvements in all OpenCL compatible devices, especially in GPUs that require thousands of threads to hide execution latencies against non-optimized version (Org). Whereas the vectorization (Vec) helps almost all OpenCL compatible devices in terms of increased hardware utilization in VLIW and SIMD designs or reduced control flow operations. Only the nVidia GeForce GTX285 revealed no performance enhancement, which was inconsistent with the profiling

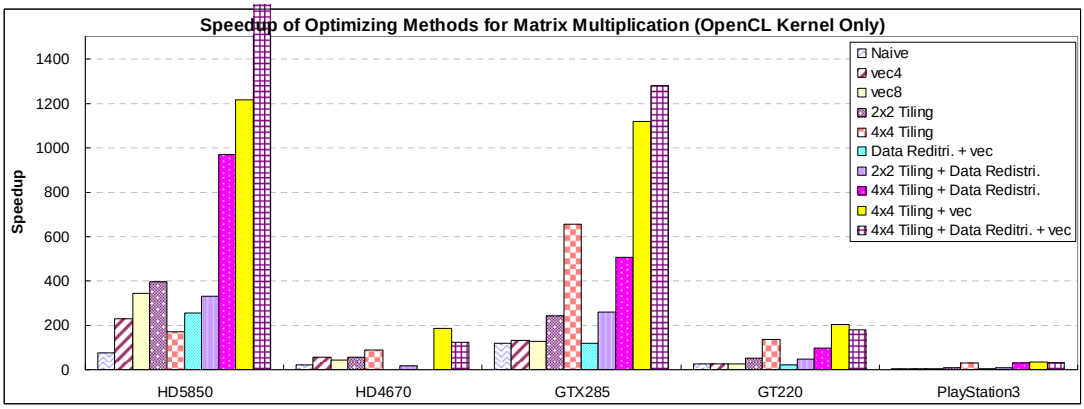
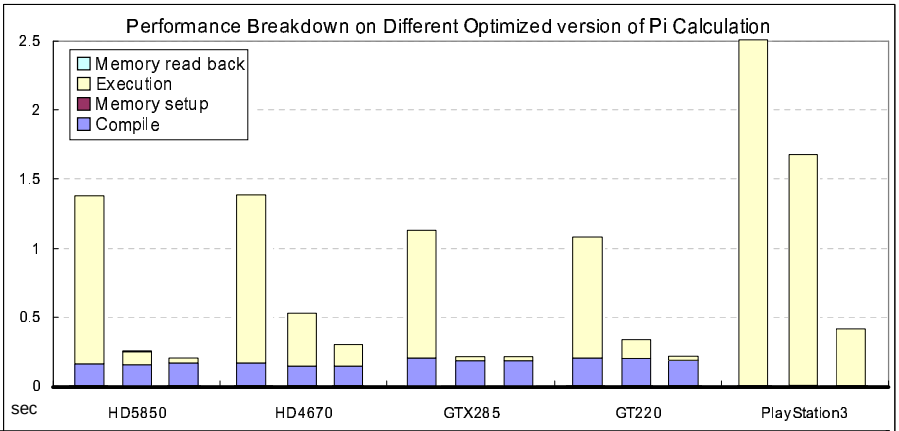


Fig. 11: Speedups of different optimizing methods on matrix multiplication. Excluded the setup and compiling time, the execution time here only adopts net execution time of OpenCL kernel.



	Org	MT	Vec	Org	MT	Vec	Org	MT	Vec	Org	MT	Vec	Org	Vec	Org
Compile	0.155	0.159	0.167	0.163	0.145	0.146	0.200	0.182	0.203	0.199	0.198	0.198	0.003	0.004	0.003
Memory setup	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Execution	1.208	0.094	0.031	1.210	0.382	0.156	0.921	0.032	0.031	0.870	0.140	0.114	2.506	1.675	0.408
Memory read back	0.001	0.001	0.000	0.001	0.002	0.000	0.001	0.000	0.000	0.001	0.000	0.005	0.001	0.000	0.000
Total	*1.364*	*0.254*	*0.198*	*1.374*	*0.528*	*0.302*	*1.122*	*0.213*	*0.234*	*1.070*	*0.338*	*0.312*	*2.510*	*1.679*	*0.412*

Fig. 13: Overall performance breakdown of optimized ”Pi calculation” with Massively Multithreading and Vectorization.

results. However, the timer may have lacked sufficient precision to measure an extremely small performance improvement. Figure 13 shows that, since this kernel is small, the overall performance of the optimized OpenCL program suffers if kernel compilation is excessively time consuming. Speedup ratio is also decreased accordingly.

4.6 Summary of Optimization of Matrix Multiplication

This subsection compares the overall speedups and performance breakdowns between different versions of optimized matrix multiplications. Comparison of overall speedups in terms of execution time only indicated that speedups were significantly lower. Moreover, the extreme optimized version with all optimization techniques obtained lower speedups with 4x4 tiling and vectorization. In other cases, the performance trend was similar in the execution time only version. Figure 14 shows the performance breakdowns. The speedup is obtained from the ratio of total execution time of OpenCL version program on various OpenCL devices over the execution time of sequential version program on host Intel Core i5 processor. Different from Fig. 11, the execution time here includes the setup and compiling time.

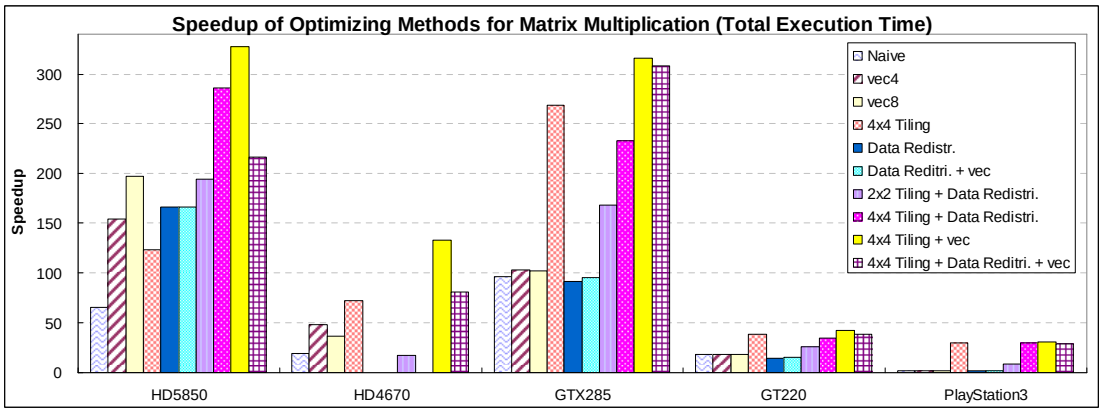


Fig. 14: Overall performance comparison of different combinations of optimized matrix multiplication. The execution time here is total execution time which includes the execution time of OpenCL kernel, setup and compiling time.

The configurations in Fig. 15 are Naive (NAV) kernel, 4x4 tiling with vectorization (TV) and 4x4 tiling with vectorization and data redistribution (TVD). Also, the GT220 matrix size is 1024x1024 instead of 2048x2048 due to hardware resource limitations. The Memory Setup time includes memory allocated for two matrices and for initializing all internal entries, so the setup time is longer than in ”Pi Calculation”. Since the number of applied optimizations increases, compilation time in runtime is also longer. Overall speedups are therefore lower than that for execution time alone.

5 Experimental Results

This section, measures the overall speedup of optimized OpenCL programs, including speedup in compilation and in memory transfer from/to host. The two above benchmarks and selected optimization results are reviewed. Then, two additional speedup benchmarks are compared between the non-optimized and optimized versions.

5.1 Experimental Setup

This study included five OpenCL compatible platforms, which are two GPUs from ATi, two GPUs from nVidia and IBM Cell processor in PlayStation3. The speedups compare with sequential version of two benchmarks run on OpenCL host device is Intel Core i5-750 for all GPUs and PPE for Cell. The speedup in different platforms was measured with timer provided by host OS with average value from multiple runs in seconds. All OpenCL kernels and host codes were identical across these five different OpenCL compatible platforms, except for vendor name string, linking library and kernel NDRange lunch parameters. Also, the OpenCL kernel compilation process in PlayStation3 was performed offline since it required tens of seconds to complete.

5.2 Pi Calculation

Figure 16 compares the non-optimized and optimized version of ”Pi calculation”. For GPUs, the original version uses only 200 work-items with 50 work-items per workgroup to solve this problem. Whereas the optimized version uses massive multithread as discussed in previous section and vectorization (with float8). Due to VLIW design in ATi and SIMD design in IBM Cell, these results show significantly improved performance compared to the non-optimized version. The speedups vary from 5 to 130 between different versions.

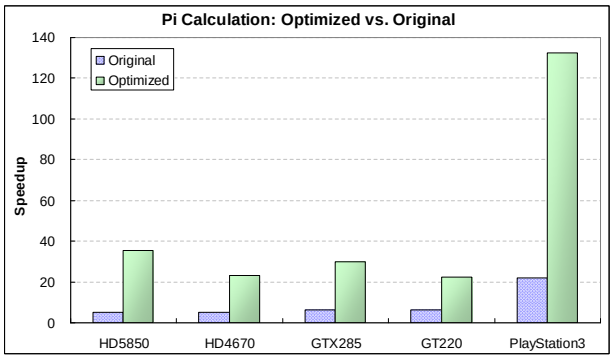
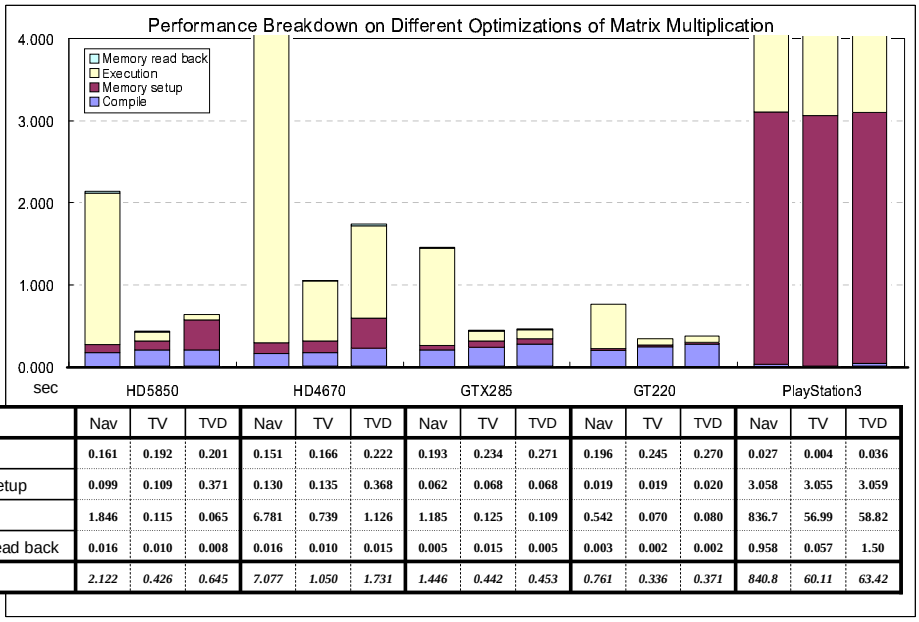


Fig. 16: Overall speedups on optimizations of ”Pi Calculation” versus its sequential version on host.



	Nav	TV	TVD	Nav	TV	TVD	Nav	TV	TVD	Nav	TV	TVD	Nav	TV	TVD
Compile	0.161	0.192	0.201	0.151	0.166	0.222	0.193	0.234	0.271	0.196	0.245	0.270	0.027	0.004	0.036
Memory setup	0.099	0.109	0.371	0.130	0.135	0.368	0.062	0.068	0.068	0.019	0.019	0.020	3.058	3.055	3.059
Execution	1.846	0.115	0.065	6.781	0.739	1.126	1.185	0.125	0.109	0.542	0.070	0.080	836.7	56.99	58.82
Memory read back	0.016	0.010	0.008	0.016	0.010	0.015	0.005	0.015	0.005	0.003	0.002	0.002	0.958	0.057	1.50
Total	*2.122*	*0.426*	*0.645*	*7.077*	*1.050*	*1.731*	*1.446*	*0.442*	*0.453*	*0.761*	*0.336*	*0.371*	*840.8*	*60.11*	*63.42*

Fig. 15: Overall performance breakdown of optimized matrix multiplication.

5.3 Matrix Multiplication

Figure 17 shows the selected results for matrix multiplication from the previous section. The original version is the naive O(N^3) algorithm whereas the optimized version with vectorization and 4x4 tiling leverages compilation overheads against net execution time. The speedups vary from 2 to 320 between different versions.

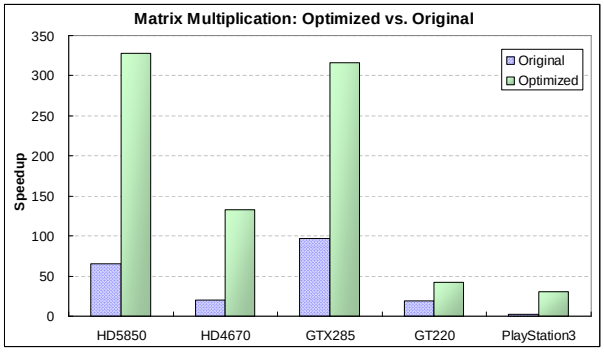


Fig. 17: Overall speedups on optimizations of ”Matrix Multiplication” versus its sequential version on host.

5.4 N-Body Simulation

The benchmark in this subsection, N-Body simulation with all-pairs N-body algorithm, is a famous physics problem. The total force on each body in the system is determined by computing each pair-wise force in the system and then summing for each body. In each component in OpenCL vector primitive in this kernel, float4 is used as different directions of velocities, and each work-item calculates one body. The number of bodies and the number of work-items are both 65,536. However, the number of work-items per workgroup is 32 for ATi HD4670, 1 for IBM Cell and 64 for the others. The original version is the one with vector primitives as velocities of different directions. The optimized version applies tiling, data redistribution with local memory, and loop unrolling two times. Figure 18 shows the results. Since not all operations in the loop can be represented by vector operator and the dependencies between instructions, the VLIW and SIMD designs do not achieve optimal performance. The packing ratio of ATi GPU is 38% for the original version and 56% for the optimized version. In PlayStation3, the compiler cannot efficiently utilize these non-vector operators to fill its SIMD lanes. Local memory mapping is also problematic. The optimized version on ATi HD4670 does not perform as well as the original due to local memory mapping problem. The speedups vary from 20 to 660 between different versions.

5.5 K-Means Clustering

A K-means clustering algorithm is commonly used for collecting digital information and for analytical tasks such as data mining. Clustering is a means of arranging n

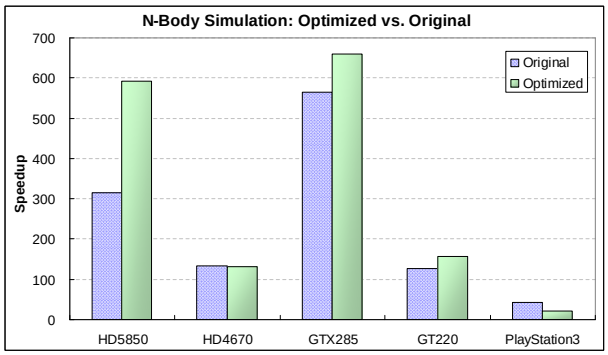


Fig. 18: Overall speedups on optimizations of "N-Body Simulation" with 65,536 bodies versus its sequential version on host.

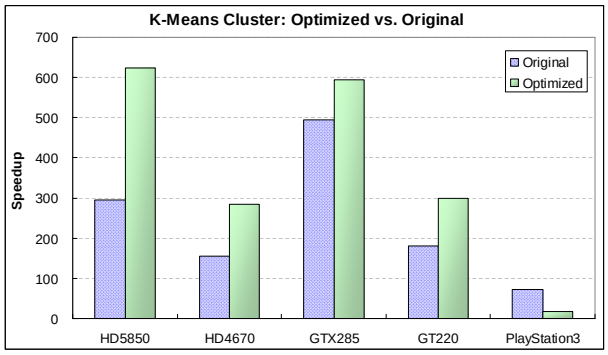


Fig. 19: Overall speedups on optimizations of "K-Means Cluster" with 1,048,576 points and 4,096 clusters versus its sequential version.

data points into k clusters where each cluster has maximal similarity as defined by an objective function. This subsection describes the conversion of a parallel K-Means clustering CUDA program [25] into OpenCL. The settings for this benchmark are 1,048,576 points and 4,096 clusters, where each work-item calculates one point against all clusters to identify the cluster with maximal similarity. The original one is the same as the CUDA version with OpenCL native function. The optimized kernel applies rectangular 1x8 tiling and unroll-and-jam with vector primitive (float8). Figure 19 shows the results. Since data access is sequential and has good locality, the speedups obtained from local memory are lagged by its longer compilation time (not shown). The optimized version can double the VLIW packing ratio from 30% to 60% for ATi GPUs and improves performance by unrolling the original loop eight times. Although the vector primitive is used, the operators in loop cannot be represented via vector operator. The speedup of PlayStation3 degrades performance because its compiler cannot arrange the code in SIMD fashion effectively. The speedups for different versions vary from 71 to 625.

5.6 Summary

This study examined four benchmarks and various optimization techniques. Full utilization of different platforms requires careful tuning of source codes. This subsection summarizes the discussion of optimization techniques their application in the various platforms shown in Table 3.

The GPUs require numerous threads to hide execution latency, so that it needs massive multithreading for hiding and prioritizing. Programmers could adopt the above discussion to adjust the total number of work-items and the size of the workgroup regardless of the N-Dimensional range used. When applying SIMD and VLIW architecture in IBM Cell and ATi Radeon GPU, the programmer must code the vector primitives in

Table 3: The summary of experimental environments.

Device / Opt. tech	ATi Radeon GPU	nVidia GeForce GPU	IBM Cell
Suggested optimization techniques (* list according to its priority)	1. Massively Multithread 2. Vectorization 3. Loop Unrolling 4. Tiling Mem Access 5. Local Share Memory	1. Massively Multithread 2. Tiling Mem Access 3. Local Share Memory 4. Loop Unrolling 5. Vectorization	1. Vectorization 2. Loop Unrolling 3. Tiling Mem Access 4. Local Share Memory 5. Massively Multithread

OpenCL which may enhance code generation and scheduling of its OpenCL compiler. Therefore, its priority is first and second IBM Cell and ATi Radeon GPU, respectively. However, its scalar design methodology limits its use in nVidia GPU. Vectorization applied on an nVidia GPU may impair performance by increasing register pressure and lowering Occupancy. Loop unrolling is helpful in almost all architectures and obtains the second and third largest performance improvements in IBM Cell, and ATi GPU respectively. Since the nVidia GPU design does not have a vector register to reduce register pressure generated from loop unrolling, unrolling may degrade performance. Therefore, it is fourth on the list for nVidia GPU. All memory access optimization techniques help to hide access latencies and requires for most of the GPUs. A massive multithread list yields the smallest performance gain in IBM Cell since it is not natively multithreaded.

This study addressed the mapping of OpenCL to hardware architectures, its execution model, and optimization techniques for different architectures. The OpenCL-compatible devices used in this study were ATi Radeon HD5850, ATi Radeon HD4670, nVidia GeForce GTX285, nVidia GeForce GT220 and PlayStation3. To use the same OpenCL kernel on different platforms, only the N-Dimensional range kernel launch processes require adjustment. The program itself need not be modified. The portability of OpenCL is a significant advantage over traditional GPGPU paradigms such as CUDA. The architectural implications for different architectures were also discussed in this study. Except for some OpenCL

runtime and device driver issues, the OpenCL kernel requires careful tuning based on underlying hardware to maximize computation ability. For ATi Radeon GPU, the VLIW design requires skillful coding of vectorization or loop unrolling to simplify the compiler and to enable instruction packing. Moreover, the theoretical performance limitation is determined by the division instruction as shown in subsections 4.3 and 4.4. The high clock rates and deep pipelined scalar processors in nVidia GeForce GPUs require huge amounts of threads to hide execution latencies, and their scalar designs limit the speed improvement obtained by vectorization. Of the OpenCL-compatible devices examined in this study, the OpenCL runtime and compiler on IBM Cell is relatively immature and revealed the worst performance. Finally, overall speedups were compared, and programming recommendations were given. Average speeds in non-optimized and optimized kernels were 24 and 430, respectively. This demonstrates the huge optimization space for OpenCL on different hardware architectures. For effective use of this open and portable programming paradigm, programmers must carefully tune OpenCL kernels to their underlying hardware characteristics. Although programming in OpenCL is harder than that in conventional OpenMP, a fine-tuned OpenCL provides a substantial performance gain in general-purpose computing on heterogeneous multicore architectures at a low cost.

6 Conclusions

This study addressed the mapping of OpenCL to hardware architectures, its execution model, and optimization techniques for different architectures. The OpenCL-compatible devices used in this study were ATi Radeon HD5850, ATi Radeon HD4670, nVidia GeForce GTX285, nVidia GeForce GT220 and PlayStation3. To use the same OpenCL kernel on different platforms, only the N-Dimensional range kernel launch processes require adjustment. The program itself need not be modified. The portability of OpenCL is a significant advantage over traditional GPGPU paradigms such as CUDA. The architectural implications for different architectures were also discussed in this study. Except for some OpenCL runtime and device driver issues, the OpenCL kernel requires careful tuning based on underlying hardware to maximize computation ability. For ATi Radeon GPU, the VLIW design requires skillful coding of vectorization or loop unrolling to simplify the compiler and to enable instruction packing. Moreover, the theoretical performance limitation is determined by the division instruction as shown in subsections 4.3 and 4.4. The high clock rates and deep pipelined scalar processors in nVidia GeForce GPUs require huge amounts of threads to hide execution latencies, and their scalar designs limit the speed improvement obtained by vectorization. Of the OpenCL-compatible devices examined in this study, the OpenCL runtime and compiler on IBM Cell is relatively immature and revealed the worst performance. Finally, overall speedups were compared, and programming recommendations were given. Average speeds in non-optimized and optimized kernels were 24 and 430, respectively. This demonstrates the huge optimization space for OpenCL on different hardware architectures. For effective use of this open and portable programming paradigm, programmers must carefully tune OpenCL kernels to their underlying hardware characteristics. Although programming in OpenCL is harder than that in conventional OpenMP, a fine-tuned OpenCL provides a substantial performance gain in general-purpose computing on heterogeneous multicore architectures at a low cost.

Acknowledgement

This work is supported in part by the National Science Council of Republic of China, Taiwan under Grant NSC 101-2221-E-033-049.

References

[1] M. McCool, S. D. Toit, T. Popa, B. Chan, and K. Moule, Proc. SIGGRAPH, 787 (2004).
[2] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan, ACM Trans. Graphics, **23**, 777 (2004).
[3] NVIDIA Corp., NVIDIA CUDA Compute Unified Device Architecture Programming Guide, Version 2.0. (2008).
[4] Khronos OpenCL Working Group, The OpenCL Specification 1.0 rev.48. (2009).
[5] J. Breitbart and C. Fohry, Proc. IEEE IPDPSW'10, 1 (2010).
[6] B. Sharma and N. Vydyanathan, IEEE IPDPSW' **10**, 1 (2010).
[7] R. Weber, A. Gothandaraman, R. Hinde and G. Peterson, IEEE Trans. Parallel and Distributed Systems, **22**, 58 (2010).
[8] S. Ryoo, C. I. Rodrigues, S. S. Stone, S. S. Baghsorkhi, S. Ueng, J. A. Stratton, W. W. Hwu, Proc. 6th IEEE/ACM International Symposium on Code Generation and Optimization, 195 (2008).
[9] S. Ryoo, C. I. Rodrigues, S. S. Baghsorkhi, S. S. Stone, D. B. Kirk, W. W. Hwu, Proc. 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 73 (2008).
[10] AMD Corp., AMD Graphics for Desktop PCs. http://www.amd.com/us/.
[11] NVIDIA Corp., NVIDIA GeForce Family. http://www.nvidia.com/.
[12] IBM Corp., Cell Broadband Engine resource center. http://www.ibm.com/developerworks/power/cell/.
[13] AMD Corp., OpenCL Programming Guide, (2010).
[14] NVIDIA Corp.,OpenCL Programming for the CUDA Architecture, (2009).
[15] OpenMP Architecture Review Board, OpenMP specification version 3.0, (2008).

[16] S. Ryoo, C. I. Rodrigues, S. S. Stone, J. A. Stratton, S. Ueng, S. S. Baghsorkhi and W. W. Hwu, Journal of Parallel and Distributed Computing, **68**, 1389 (2008).
[17] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer and K. Skadron, "Journal of Parallel and Distributed Computing, **68**, 1370 (2008).
[18] AMD Corp.,R600/R700/Evergreen Assembly Language Format, (2009).
[19] AMD Corp., Stream Kernel Analyzer. http://developer.amd.com.
[20] AMD Corp., ATi Stream Profiler. http://developer.amd.com.
[21] NVIDIA Corp., NVIDIA CUDA Visual Profiler. http://developer.download.nvidia.com.
[22] AMD Corp., ATI Stream Software Development Kit (SDK). http://developer.amd.com/.
[23] E. Lindholm, J. Nickolls, S. Oberman and J. Montrym, IEEE Micro Magazine, **28**, 39 (2008).
[24] AMD Corp., R600-Family Instruction Set Architecture, (2007).
[25] R. Farivar, D. Rebolledo, E. Chan and R. Campbell, Proc. 2008 International Conference on Parallel and Distributed Processing Techniques and Applications, 663 (2008).
[26] R. Wang, N. Sha, B. Gu, X. Xu, The Characterizations of Discrete Life Distribution Class with Relation to Geometric Distribution, Journal of Statistics Applications & Probability, **2**, 91-101 (2013).
[27] S. Y. Alshareeda, PRM Mitigation Scheme for the Nodes Failure Effect in the WSNs CGD Protocol, International Journal of Computing and Digital Systems, **2**, 29-37 (2013).
[28] A. Lee, I. Ra, Applied Mathematics & Information Sciences, **6**, 271 (2012).
[29] J. S. Zhang, A. X. Chen, Review on quantum discord of bipartite and multipartite systems, Quantum Physics Letters, **1**, 69-77 (2012).
[30] M. R. Girgis, T. M. Mahmoud, H. F. Abd El-Hameed, Z. M. El-Saghier, Routing and Capacity Assignment Problem in Computer Networks Using Genetic Algorithm, Information Sciences Letters, **2**, 13-25 (2013).

Slo-Li Chu received his PhD degree in Electrical Engineering from National Sun Yat-sen University in 2002. From August 1998 to 1999, he visited Department of Computer Science, University of Illinois at Urbana-Champaign for one year. He is currently an assistant professor of Department of Information and Computer Engineering, Chung Yuan Christian University, Taiwan. His research interests include computer architectures, parallelizing compilers, system level modeling, system-on-chip design, and GPU architectures.

Chih-Chieh Hsiao received his BS degree in Information and Computer Engineering from Chung Yuan Christian University, Taiwan, in 2007, and the MS degree in Computer Science and Information Engineering from National Chiao Tung University, Taiwan, in 2009. He is currently pursuing for his PhD degree in Electronic Engineering at Chung Yuan Christian University, Taiwan. His research interests include computer architectures, parallel programming, and GPU architectures.