PerfExpert and MACPO: Which code segments should (not) be ported to MIC?

Ashay Rane
Texas Advanced Computing Center
The University of Texas at Austin
ashay.rane@tacc.utexas.edu

James Browne
Department of Computer Science
The University of Texas at Austin
browne@cs.utexas.edu

Lars Koesterke
Texas Advanced Computing Center
The University of Texas at Austin
lars@tacc.utexas.edu

***Abstract*—Accelerators like Graphics Processing Units (GPUs) or coprocessors like Intel's MIC (Many Integrated Core) provide means to exploit large-scale SIMT/SIMD parallelism in applications. Tools for converting CPU code to code for accelerators/coprocessors are available. Application developers could quickly exploit these accelerators/coprocessors with modest effort if they could identify the code segments that are suited for effective and efficient execution on these architectures without algorithmic changes. However, such an analysis requires knowledge of models of parallelism and execution behavior of the code segments on multicore CPUs. In this work, we present a semi-automated, tool-supported process of identifying the code segments that will run efficiently on accelerator chips such as GPUs and coprocessors such as Intel MICs. Results on production codes demonstrate reliable separation between code segments that can be directly converted to execute efficiently on the MIC versus those that cannot.**

***Keywords*-performance; accelerators; GPU; MIC**

I. Introduction

The compute nodes of the emerging generation of high performance computers are expected to include both multicore chips which execute asynchronous SPMD/MIMD parallelism (but usually include vector operations), and accelerators (GPUs) or coprocessors such as Intel's Many Integrated Core (MIC) architecture which implement synchronous SIMT/SIMD/Vector/Streaming parallelism, respectively. These coprocessors are a low-cost, low-power means of attaining large-scale parallelism. Source-to-source transformation tools for mapping CPU code for accelerator execution are available. However, in a large application the identification of the code segments that can be translated for efficient execution on accelerators/coprocessors requires expert knowledge of algorithms, architectures, compilers and the program structure which many application developers may not possess. A search of the literature found almost no methods or tools supporting identification of those code segments which can directly benefit from execution on an SIMT/SIMD accelerator or coprocessor.

The principal contributions of this paper are specification of the characteristics of code segments which can be directly compiled for efficient execution on accelerator/coprocessor chips and a systematic process for obtaining these characteristics, along with identifying and ranking of those code segments in a large application optimized for multicore based compute nodes which can (and cannot) be directly mapped for effective execution on SIMT/SIMD accelerator such as the Intel MIC. The measurements and analyses required for this process can be obtained using tools which are available for multicore chips and multichip nodes of high performance computers. Of course, in some cases it will be necessary to reformulate the computations of important code segments using different algorithms or manually recode in an accelerator language. While we do not directly consider these cases, those code segments which must be reformulated or recoded are identifiable from the analyses presented here.

The characteristics required of the code segments which can be directly mapped for most effective SIMT/SIMD execution include: scalable SPMD/MIMD and/or streaming/vector parallelism, computational intensity, regular strides through the data structures of the code segment and high data reuse factor. The characteristics which render code segments unsuitable for direct compilation for accelerator execution include an appreciable rate of TLB misses, a high rate of branch misses, cache conflicts across cores and irregular access strides for important data structures.

The contents of the paper are: a detailed description of the code segment identification process, three illustrations of the process, one being a simple example and the other two being full scale production codes. The effectiveness of the mappings is demonstrated with execution on Intel MIC chips. Some of the advantages in utilizing the Intel software for mapping OpenMP code directly for MIC execution are utilized for the Moil [1] code.

II. Related Research

The entire process for transforming large applications to execute on accelerators/coprocessors has multiple steps: identifying and characterizing the code segments which can be directly mapped for accelerator execution, implementing the accelerator executable, either by compiler or by manually recoding and optimizing the code on the accelerator. While this paper contributes only to the first step, it seemed worthwhile to document that methods and tools for mapping CPU code to accelerator code are in place.

Rane, Sardeshpande and Browne [2] presented an earlier and less complete version of the process presented in this

paper as a poster at SC11. The only other reference to a process for identification and characterization of code segments for effective SIMT/SIMD execution that we have found in the literature is a recommendation on the CloudBlue web site [3] to use `gprof` to determine the execution time of code segments. The other most closely related previous work is GProphecy [4] which estimates the performance of a skeletonized piece of CPU code which has been targeted for GPU execution from an analytical model.

Existing tools such as the PGI compiler [5], C-to-CUDA for affine programs [6], OpenMPC [7], Mint [8] and Absoft [9] can produce GPU code from an annotated legacy code (some both C and Fortran). The Intel C, C++ and Fortran compilers can directly compile OpenMP/MIC annotated code for Intel MIC chips. The PGI Accelerator [10] also compiles OpenMP annotated code to OpenCL and CUDA.

Performance models increasingly have been used for application tuning over complex or large scale systems. These models target performance over a cluster or a heterogeneous platform, with a focus on the modeling and optimization of communication and scheduling among nodes. One such example is MDR [11], which models heterogeneous platforms.

III. Code segment identification process

We first summarize briefly the PerfExpert [12], [13] and MACPO [14] tools we use obtain the measurements and analyses needed to implement this process. PerfExpert is a widely used performance optimization system for multicore chips and multicore nodes of high performance computers. PerfExpert gathers all of the performance counters needed to identify the code segments in an application which have performance bottlenecks due to poor usage patterns for memory and/or computational resources, determines the specific causes of each bottleneck and then recommends source to source transformations for optimizing each bottlenecked code segment or recommends compiler switches which will avoid occurrence of the bottlenecks. Mapping for execution on an accelerator is an additional (but very important) optimization which can be recommended for a code segment. The analyses generated by PerfExpert include: computational intensity by code segment and a decomposition of the CPI for a code segment into contributions (Local Cycles per Second LCPI) from each level of memory (L1, L2, L3, chip local memory, TLB and NUMA) and from different classes of instructions such as branches. These contributions are scaled for the architecture upon which they are executing so that “good” or “poor” values are obvious. The MACPO tool generates memory traces of the important data structures by code segment. These memory traces are processed to obtain access strides and data reuse factors for each data structure separately by code segment, the average latency of accesses for each important data structure with attribution to different levels of cache and memory from L1 caches to NUMA accesses.

1. **Optimization for Multicore Chips and Multichip Node**: The first step is to thoroughly optimize the code for the multicore chips/multichip nodes which implement the asynchronous SPMD/MIMD parallelism. The rationale is that optimization for multicore chips and the multichip nodes maximizes asynchronous and streaming/parallelism which can be mapped to the synchronous SIMT/SIMD parallelism of the accelerators and coprocessors. For this step we primarily use the PerfExpert performance optimization system.

2. **Identify time-consuming code segments and characterize their execution characteristics using PerfExpert, MACPO and on-chip scaling behavior on the optimized code**: All of the data and analyses which are needed are generated by PerfExpert and MACPO.

3. **Eliminate code segments which cannot be directly mapped for effective SIMT/SIMD execution**: Code segments with substantial contributions to LCPI [12] from TLB misses or branches and with cache conflicts across cores or accesses with irregular strides for the important data structures of the code segment can be eliminated from consideration. These code segments cannot be directly mapped for effective SIMT/SIMD execution.

4. **Characterize those code segments which may be directly mapped for effective SIMT/SIMD execution**: The metrics for ranking probable speedup using mapping for accelerator/coprocessor execution as the optimization are: computational intensity, scalability of the SIMD/MIMD parallelism, presence of streaming/vector parallelism, regular access strides for important data structures and data reuse factors for the important data structures.

5. **Rank the code segments identified in Step 4 as directly mappable for their potential performance gain by SIMT/SIMD execution on a MIC by using the metrics generated in the previous step**: This ranking can be done by sorting on one or more of the metrics including total time to execute on the multicore chips or by generating a predictive formula using previously ported codes.

6. **Refactor the application to put mappable code segments with common data close together in execution order**: This step is manual at this time but it can be automated with use of compiler generated data flow graphs.

7. **Use compilers to implement mappings**: The step is dependent upon the accelerator chip. For the MIC chips used in this study, we used the Intel compiler to transform

Table I
ELIMINATION METRICS FOR MOIL

Code segment	TLB	Branch	FLOPs	Conflicts
Non-water calc.	0	0.1	19%	0
Water calc.	0	0	23%	0
Convolution	0	0.2	17%	0
passf4_dp()	0	0	46%	0
passb4_dp()	0	0	45%	0
grad_sum_dp()	0.4	0	7%	0
PME calc.	0.2	0.2	0%	0

Table II
ELIMINATION METRICS FOR BIO

Code segment	TLB	Branch	FLOPs	Conflicts
Kernel	0	0.1	46%	0

Table III
ELIMINATION METRICS FOR HEAT3D

Code segment	TLB	Branch	FLOPs	Conflicts
Kernel	0	0	5%	0

Table IV
RANKING METRICS FOR MOIL

Code	Speedup	SSE inst.	Stride score	Reuse dist
Non-water calc.	5.8	19%	1	12
Water calc.	8	20%	0.9	10
Convolution	7.8	20%	1	5
passf4_dp()	8.3	37%	0.7	6
passb4_dp()	8	32%	0.4	6

Table V
RANKING METRICS FOR BIO

Code	Speedup	SSE inst.	Stride score	Reuse dist
Kernel	5.37	25%	0.95	5

OpenMP code into MIC executable.

IV. RESULTS

In this section, we present the results of applying the process described in the previous section to three application codes – a 3D heat transfer application (Heat3D), an application [15] (Bio) to evaluate the correlation between pairs of plant genes and observables like plant height, kernel size, etc. and a molecular dynamics and modeling application (MOIL) [1]. To validate that optimization for multicore execution does prepare the codes for SIMT/SIMD execution we applied the identification process to the Bio and Heat3D codes to unoptimized versions. We were not able to find any code segments with metrics suggesting efficient execution on the MIC in these unoptimized codes. However, as will be seen, for the optimized versions, the important code segments in both codes became highly scalable on the MIC. Optimizations applied to these codes centered around tuning memory performance.

Tables I, II and III show metrics generated using PerfExpert and MACPO for eliminating code segments from consideration that are not suited for execution on accelerators. The tables show metric values only for those code segments which consume a significant fraction of the execution time of the application. We use the Data Translation Lookaside Buffer (DTLB) & Branch LCPIs and the code's Floating Point Operations per second (FLOPs) from PerfExpert. The FLOPs value is a percentage of the practically obtainable maximum that can be achieved on the given architecture. Tables IV, V and VI show metrics used for ranking the code segments for suitability for accelerator/coprocessor execution. For each code segment that was not eliminated in the prior step, we measure the strong-scaled speedup from 1 to 6 threads on a six-core Westmere processor. We restrict our scaling measurements to intra-chip scaling to avoid NUMA and cross-package communication, which would complicate the analysis. We take into account the computational intensity (FLOPS) and percentage of executed SSE instructions to understand the degree of streaming parallelism present in the code segment. The "stride score", derived from data structure analysis using MACPO, is a metric of the uniformity in the strides – i.e. higher diversity in the strides causes the stride metric to be low. The score is calculated by summing the cubes of the normalized stride counts. The score itself is normalized to vary between 0 and 1. Reuse distance values are used to determine locality properties of the source code. MACPO produces reuse distance measurements at the granularity of cache lines (usually 64 bytes).

The ranking metrics demonstrate that the computational

Table VI
RANKING METRICS FOR HEAT3D

Code	Speedup	SSE inst.	Stride score	Reuse dist
Kernel	5.63	3%	0.83	8

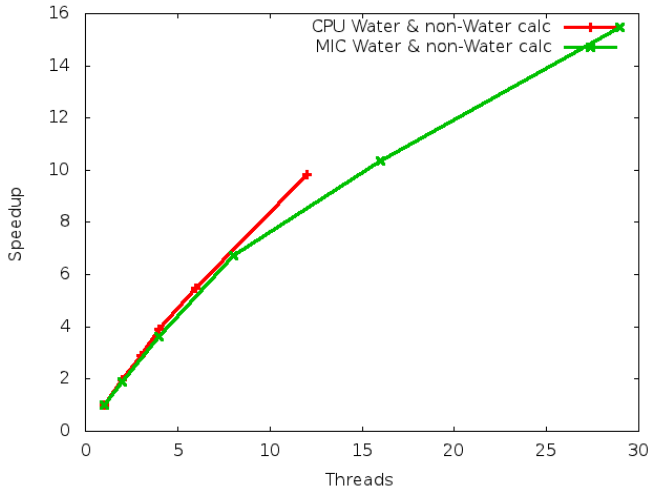

Figure 1: Speedup comparison plot for Water and non-Water calculations in MOIL

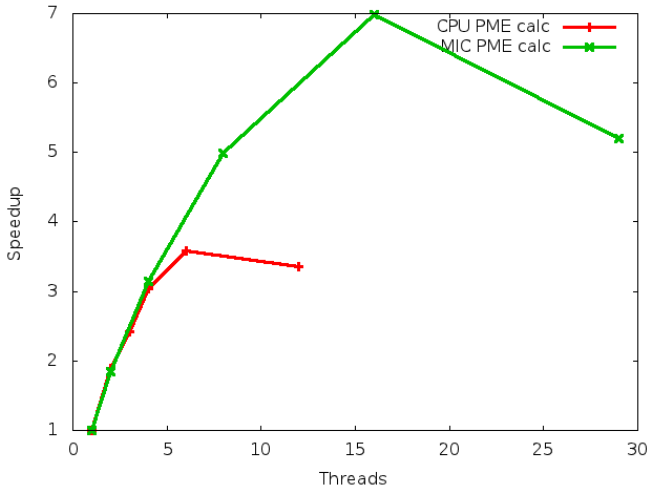

Figure 2: Speedup comparison plot for PME calculations in MOIL

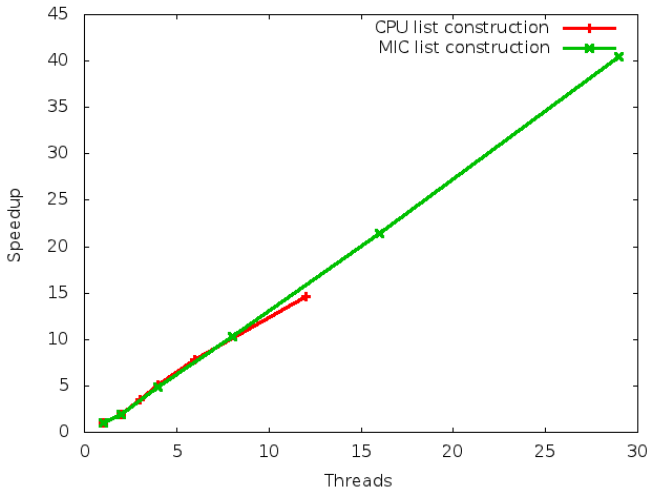

Figure 3: Speedup comparison plot for list construction in MOIL

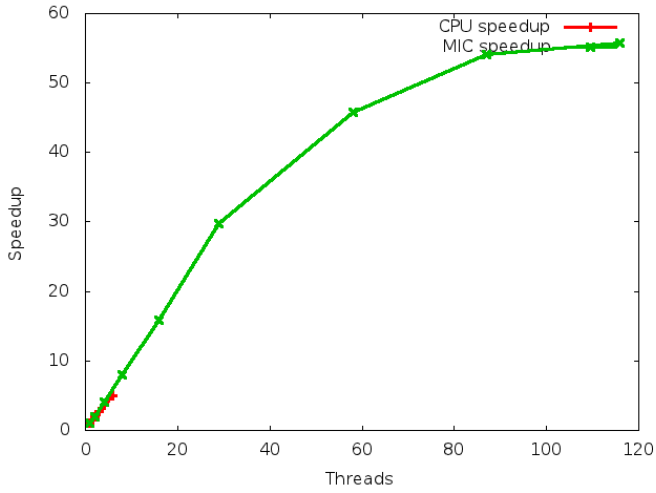

Figure 4: Speedup comparison plot for Bio

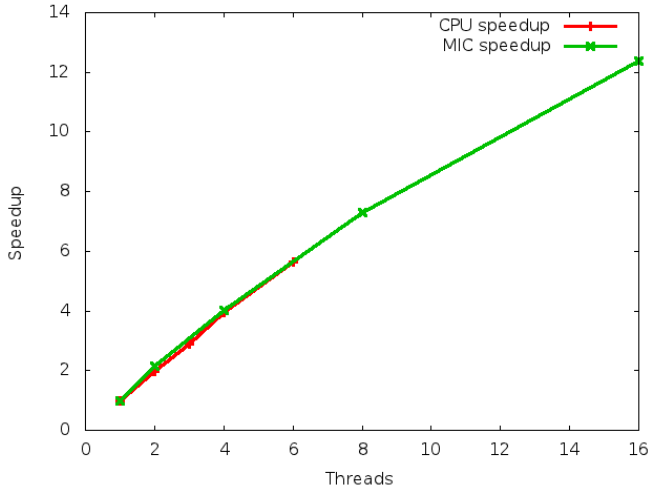

Figure 5: Speedup comparison plot for Heat3D

kernel of the Bio code is extremely well suited for running on the MIC. Metrics for the Heat3D code show good memory access patterns, however the FLOP count is low, reducing the confidence in its ability to run well on the MIC. A possible optimization to make this code run even more efficiently on the MIC would be to discover means of exploiting wider streaming (SSE) parallelism to increase the number of FLOPs. For the MOIL code, we see that Water and non-Water calculations and the convolution operation would likely run well on the MIC. Although the `passb4_dp()` and `passf4_dp()` functions scale well on the CPU, their scalability on the MIC is doubtful because of their diverse strides. Most notably, the PME calculations and `grad_sum_dp()` routine were eliminated from further consideration because of their poor TLB and branch characteristics.

Based on this analysis, we ran the applications on the MIC and measured the speedup of the code sections on the MIC hardware. Speedup comparison plots are shown in Figures 1-3, 4 and 5. The comparisons show that our analysis is generally correct as one can see that the code sections recommended by our process scale very well on the actual hardware. Also, certain code sections such as the PME calculations in the MOIL code, which was not recommended for running on the accelerator scales quite poorly on the MIC.

While indicating that these recommendations are generally correct, the comparisons also showed avenues for improvement. For instance, the list construction function in the MOIL code (`GetCPUParNBNbrListWater()`) (that generates data for the compute-intensive functions recommended by our process) was not recommended to run

on the MIC because it consumes less than 1% of the total time. However, not only should this function to be ported to the accelerator to minimize data movement between CPU and the MIC, it also scales almost perfectly as we observed from measurements on the MIC hardware (see Figure 3). This suggests including data flow between code segments into the analysis.

V. Summary and Conclusion

We have identified the characteristics and properties of those code segment of large applications (optimized for multicore chip execution) which can be converted directly to accelerator/coprocessor code that will execute efficiently on SIMT/SIMD parallel accelerators. We have identified methods and tools for obtaining these characteristics and developed a systematic process for identifying and ranking code segments for SIMT/SIMD execution and validated the process on several applications. We also validate the correctness of the identification process on several substantial application codes. This work thus provides method and tool support for the heretofore unsupported step in migrating application codes for effective use of SIMT/SIMD accelerators/coprocessors.

VI. Future work

The process for identifying and characterizing code segments still has several manual steps. Integration of the measurements and analyses of PerfExpert and MACPO is still manual. We also plan to incorporate recommendations for optimizations by mapping to SIMT/SIMD accelerators/coprocessors into AutoSCOPE and to, when possible, generate code annotations to guide compiler source to source transformations from CPU to accelerator code. We also plan to add data flows between code segments into the analysis to enable grouping of code segments which share data structures. Extension of PerfExpert and MACPO for optimization on SIMT/SIMD accelerators/coprocessors is a long term goal.

Acknowledgments

This project is funded in part by the National Science Foundation under OCI award #0622780.

References

[1] R. Elber, A. Roitberg, C. Simmerling, R. Goldstein, H. Li, G. Verkhivker, C. Keasar, J. Zhang, and A. Ulitsky, "Moil - a Program for Simulations of Macromolecules." *Computer Physics Communications*, vol. 91, no. 1-3, pp. 159–189, 1995. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/001046559500047J

[2] A. Rane, S. Sardeshpande, and J. Browne, "Determining Code Segments that Can Benefit from Execution on GPUs," in *2011 ACM/IEEE Conf. on Supercomputing*, 2011.

[3] CloudBlue, "Porting C code to GPU nodes." [Online]. Available: https://mss.ccs.uky.edu/wiki/tiki-index.php?page=Porting+C+code+to+GPU+nodes

[4] J. Meng, V. A. Morozov, K. Kumaran, V. Vishwanath, and T. D. Uram, "GROPHECY : GPU Performance Projection from CPU Code Skeletons," *Architecture*, p. 1, 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=2063384.2063402

[5] PGI, "PGI CUDA Fortran compiler." [Online]. Available: http://www.pgroup.com/resources/cudafortran.htm

[6] M. M. Baskaran, J. Ramanujam, and P. Sadayappan, "Automatic C-to-CUDA Code Generation for Affine Programs," *Compiler Construction*, vol. 6011, pp. 244–263, 2010. [Online]. Available: http://www.springerlink.com/content/8h69403625839283/

[7] S. Lee and R. Eigenmann, "OpenMPC : Extended OpenMP Programming and Tuning for GPUs," *Transformation*, no. November, pp. 1–11, 2010. [Online]. Available: http://portal.acm.org/citation.cfm?id=1884674

[8] D. Unat and S. B. Baden, "Mint : Realizing CUDA performance in 3D Stencil Methods with Annotated C," *System*, pp. 214–224, 2011. [Online]. Available: http://cseweb.ucsd.edu/groups/hpcl/scg/papers/2011/mint-unat-ics11.pdf

[9] Absoft, "Absoft Pro Fortran." [Online]. Available: http://gpuscience.com/news/new-absoft-pro-fortran-2012-supports-gpu-hmpp-preprocessor-and-hpc-libraries/

[10] PGI, "PGI Accelerator Compilers." [Online]. Available: http://www.pgroup.com/resources/accel.htm

[11] J. Pienaar, A. Raghunathan, and S. Chakradhar, "MDR: Performance Model Driven Runtime for Heterogeneous Parallel Platforms," in *Proceedings of the 25th International Conference on Supercomputing*, ser. ICS'11. ACM, 2011.

[12] M. Burtscher, B.-D. Kim, J. Diamond, J. Mccalpin, L. Koesterke, and J. Browne, "PerfExpert : An Easy-to-Use Performance Diagnosis Tool for HPC Applications," in *Computer*. IEEE, 2010, pp. 1–11.

[13] O. Sopeju, M. Burtscher, A. Rane, and J. Browne, "AutoSCOPE: Automatic Suggestions for Code Optimizations using PerfExpert," *2011 International Conference on Parallel and Distributed Processing Techniques and Applications*, 2011.

[14] A. Rane and J. Browne, "Performance Optimization of Data Structures Using Memory Access Characterization," in *2011 IEEE International Conference on Cluster Computing*. IEEE, Sep. 2011, pp. 570–574.

[15] L. Koesterke, D. Stanzione, M. Vaughn, S. Welch, W. Kusnierczyk, J. Wang, C.-T. Yeh, D. Nettleton, and P. Schnable, "An Efficient And Scalable Implementation of SNP-pair Interaction Testing for Genetic Association Studies," in *HiCOMB*, 2011. [Online]. Available: http://www.hicomb.org/proceedings.html