Mapping of Applications to Heterogeneous Multi-cores Based on Micro-architecture Independent Characteristics

Jian Chen, Nidhi Nayyar and Lizy K. John
Department of Electrical and Computer Engineering
The University of Texas at Austin
{chenjian, nidhin}@mail.utexas.edu,ljohn@ece.utexas.edu

Abstract

Heterogeneous multi-core processor is demonstrated to be more efficient than its homogeneous counterpart due to its ability to meet different resource requirements of the applications. One of the challenges of designing a heterogeneous multi-core is how to schedule programs to the core that can execute them most efficiently. This paper presents a method to map application to its optimum core by analyzing the micro-architecture independent characteristics of that application. The proposed method exploits data/instruction reuse distance and true dependency distance to derive switching gains and ranks for each configuration, and maps the application to the optimum core accordingly. The experiment result shows that four out of the five representative programs under study are mapped to the optimum core in terms of energy-delay product. This study opens the possibility to design a more intelligent dynamic program switching mechanism than the current trial-and-error approach.

I. Introduction

Multi-core architectures are becoming an attractive design alternative due to the capability of achieving high instruction throughput as well as the flexibility to meet specific performance and power constraints. The existing multi-core architectures can be divided into two categories, i.e., homogeneous multi-core processor and heterogeneous multi-core processor. A homogeneous multi-core processor can be implemented by duplicating multiple copies of the same core, which is desirable from the perspective of design and validation complexity [1]. However, a homogeneous multi-core system does not give any credit to the fact that the different programs have very different resource requirements during the execution. The heterogeneous multi-core processor, on the contrary, is able to accommodate different resource demands of applications by scheduling programs to the core that can execute them most efficiently, hence, is more efficient than its homogeneous counterpart in terms of power consumption and area cost [1][2].

In order to exploit the core diversity in heterogeneous multi-core processor, programs have to be scheduled or mapped to the proper cores that are most suitable for the execution. Prior research mainly focuses on dynamic core selection based on sampling the behavior of neighboring or all cores during the switching intervals [3]. Although this dynamic method can identify program phase changes during runtime and make corresponding core switching, it gives no insight into the relationship between inherent program behavior and the corresponding resource requirements. Such relationship is important in a sense that it can not only help mapping applications statically according to the off-line profiling but also potentially lead to more intelligent dynamic core selection algorithms.

This is the reason that motivates us to understand how micro-architecture independent characteristics shape and modulate the resource demands of programs. As one step towards this direction, this paper presents a method to map applications to the proper cores statically based on micro-architecture independent characteristics. The proposed method uses data/instruction reuse distance and register dependency distance to characterize the data/instruction working set sizes and the ILP (Instruction Level Parallelism) in the program. It introduces the concept of switching gain to quantify the benefit and cost of switching from one core to another. A ranking system based on the gains is employed to determine the core configuration that can best fit runtime resource requirements of the program. The experiment result shows that four out of the five representative programs under this study are mapped to the optimum core in terms of energy-delay product. This result demonstrates the possibility to correlate micro-architecture independent characteristics of programs with optimum core configuration in a heterogeneous multi-core system.

The rest of the paper is organized as follows: Section II presents the overall framework for the proposed static application mapping. Section III describes the experiment environment used in this research. Section IV gives detailed description of the mapping heuristics

as well as the evaluation results of the proposed mapping policies. Section V concludes the paper and points out the future work.

II. Framework for Application Mapping

Figure 1 shows the framework for application mapping based on micro-architecture independent characteristics. The framework starts with an application space, which is a collection of different programs that will be running on the multi-core processor. These programs are profiled according to a set of micro-architecture independent metrics, followed by PCA (Principal Component Analysis) and clustering. The immediate result of these steps is that application space is partitioned into several clusters with one representative program for each cluster. The underlying rationale for this step is two-fold. First, the representative programs constitute a desirable subset of the application space for the demonstration of the mapping algorithm because these representative programs are different by nature and are more likely to be mapped to different cores. Second, the reduced number of programs under study can significantly accelerate the time-consuming validation process without losing generality. The mapping heuristic then takes the micro-architecture independent characteristics of these representative programs as well as the configurations of heterogeneous cores to produce the program-core mapping table. The step highlighted with dash line is only used for the validation of the mapping algorithm, hence is not the integral part of the mapping process. It should be noted that the framework only targets at single-ISA heterogeneous multi-core system [2], though it can be further extended to multi-ISA systems. In addition, this framework assumes programs running independently on different cores, and there is no communication and memory sharing between programs. This assumption may not be true in real world multi-core environment, yet it helps to identify the essential relationship between programs and hardware without being interfered with the side effects of the program communications.

The following subsections elaborate some aspects the proposed framework, including the details of microarchitecture-independent characteristics, principle component analysis and clustering.

A. Microarchitecture Independent Metrics

Microarchitecture independent metrics give the opportunity to understanding the inherent program characteristics isolated from features of particular microarchitectural components [4]. These program characteristics can be measured effectively through instrumentation which is substantially faster than simulation. This paper employs two different types of the metrics, i.e., register RAW (Read-after-Write) dependency distance to capture the instruction level parallelism and data/instruction reuse distance to capture the working set sizes. These two metrics are among the factors that have the most significant impact on the hardware resource requirements of the program, therefore have the strongest correlations with the core configurations.

RAW Dependency Distance: Dependency distance is defined as the total number of instructions in the dynamic instruction stream between the producer and the first consumer of a register instance [4]. Only true dependency distance, i.e., RAW dependency distance is profiled because false dependencies can be easily removed by register renaming and cannot reveal the

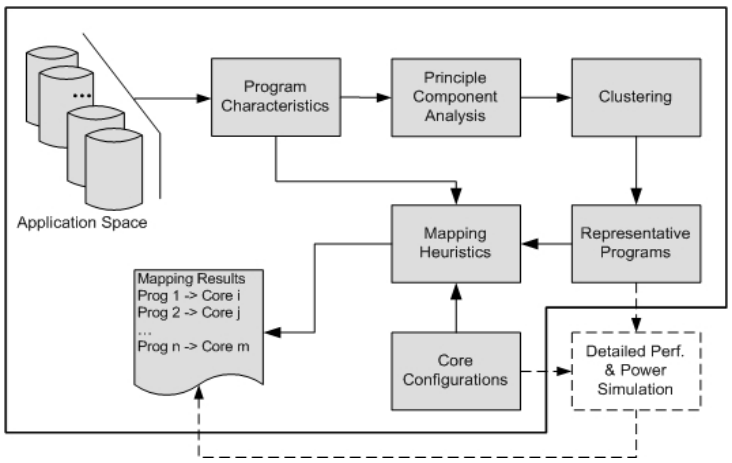


Figure 1. Framework for Application Mapping

degree of concurrency existed in the program. Since the integer RAW dependency distance and the floating-point one.

Data/Instruction Reuse Distance: The use of average memory reuse distance to characterize data temporal locality is proposed in [4]. The reuse distance is defined as the number of memory accesses between two consecutive memory accesses to the same block address. The average reuse distance can be calculated according to the distribution of reuse distance with the window size of 16, 64, 256, and 4096. The size of the block that the address indexed into can vary from 16 byte to 256 byte. One may want to sweep through all L1 block sizes that are implemented in the target multi-core processor. Table I summaries Microarchitecture independent metrics that are used in the proposed framework for program profiling and clustering.

Table I. Micro-architecture Independent Metrics used for Characterizing Benchmarks.

No.	Category	Characteristics
1	Integer RAW Dependency Distance	Distance of 1
2		Distance upto 2
3		Distance upto 4
4		Distance upto 8
5		Distance upto 16
6		Distance upto 32
7		Distance larger than 32
8	Floating Point RAW Dependency Distance	Distance of 1
9		Distance upto 2
10		Distance upto 4
11		Distance upto 8
12		Distance upto 16
13		Distance upto 32
14		Distance larger than 32
15	Data temporal Locality	Average distance for window size 16
16		Average distance for window size 64
17		Average distance for window size 256
18		Average distance for window size 4096
19	Instruction temporal Locality	Average distance for window size 16
20		Average distance for window size 64
21		Average distance for window size 256
22		Average distance for window size 4096

B. Principal Component Analysis & Clustering

Theoretically, a program can be mapped to a particular core by inspecting its micro-architecture independent program characteristics. However, in order to measure the quality of the mapping, detailed micro-architecture level simulations are needed for validation. The simulation time constraints prevent the mapping-and-validation process on a program-by-program basis from being practical. Therefore, this framework employs cluster analysis to subset the application space into several clusters, and uses the representative programs from these clusters as the major programs for mapping.

In order to cluster programs, this framework applies principal component analysis to transform the raw data to an orthogonal space with reduced dimensionality. PCA achieves dimensionality reduction by removing principal component with lowest variance and retaining the most important factors that contribute to the covariance among different programs. This framework employs 95% criterion for data reduction, that is, 95% of the total variance should be explained by the retained principal components.

The clustering is based on the Euclidean distance between different programs calculated in the space spanned with the reduced principal components. Programs that are close to each other in application space have similar characteristics and tend to be mapped to the same core, thus, should be grouped together. The immediate benefit of clustering is the reduction of the number of programs under analysis because other programs in the same cluster may follow the mapping of the representative program. This framework employs K-means clustering technique. The optimum number of clusters for K-means is determined by BIC (Bayesian Information Criterion) [6].

III. Experiment Setup

The application space of the experiment is composed of a broad range of benchmark suites, including SPEC CPU2000INT, SPEC CPU2000FP and MediaBench. These programs are compiled to Alpha-ISA. We modified SimProfile from Simplescalar tool set [5] to instrument programs and collect the above mentioned characteristics. To reduce the time for profiling, each SPEC2000 program is profiled at its single Simpoint interval with 100 million instructions [6] instead of the entire run of the program. We use STATISTICA to perform principal component analysis on the profiled program characteristics. The programs are then grouped into 5 clusters with K-means cluster algorithm. The number of clusters is determined using BIC with the tools provided in Simpoint tool suite. Table II shows the clustering results. The highlighted program for each cluster is the program that has shortest Euclidean distance toward the center of the cluster, thus is

the representative program for that cluster. These representative programs are chosen as the target applications to be mapped to different cores.

Table II. Clusters based on overall program characteristics

Cluster 1	**bzip2**, parser, adpcm, gcc, gzip, perlbmk, twolf, bzip2, vpr, applu, fm3d
Cluster 2	**cjpeg**, epic, unepic, mesa_med, mpeg2decode, mpeg2encode, rasta, encode, decode
Cluster 3	**equake**, djepg, lucas
Cluster 4	**vortex**, ghostscript, crafty, mesa, eon
Cluster 5	**art,** mcf, swim

Our hypothetical single-ISA heterogeneous multi-core processor has four different cores. The configurations of these cores should be able to demonstrate enough heterogeneity so that the mapping of an application to different cores could yield noticeable difference in terms of performance and energy consumption. This paper focuses on two key aspects of processor configurations, namely, issue width and cache size. These two aspects are consistence with the profiled micro-architecture independent characteristics, and are believed to define the performance of processors at large. Table III gives the detailed information of these four core configurations. Note that each core has a private L1 and L2 cache.

Table III. Core Configurations for Multi-core Processor

Configurations	**Details**
Configuration 1	In-order, single-issue, 2lev(1k), 8k 2-way d-cache 128byte, 4k 2-way i-cache 64byte, 32k L2 cache
Configuration 2	Out-of-order, 2-issue, 2lev(1k), 16k 4-way d-cache 128byte, 8k 2-way i-cache 64byte, 64k L2 cache
Configuration 3	Out-of-order, 4-issue, 2lev(4k), 32k 4-way d-cache 128byte, 16k 2-way i-cache 64byte, 128k L2 cache
Configuration 4	Out-of-order, 8-issue, 2lev(4k), 64k 4-way d-cache 128byte, 32k 2-way i-cache 64byte, 512k L2 cache

In order to evaluate the quality of the proposed application mapping, Wattch-1.02 [7] is employed to perform detailed power and performance simulations. Each of the five representative programs is simulated with 4 different configurations. These programs except for MediaBench have a huge amount of dynamic instruction count, which requires extremely long time to simulate. Therefore, for each SPEC benchmark, one single Simpoint interval with 100 million instructions is used for detailed micro-architecture level simulation, which is consistent with the one used in profiling. The warm-up effect in 100 million instruction interval is relatively small, hence is ignored.

IV. Mapping Heuristics and Evaluation

The mapping heuristic proposed in this paper exploits two aspects of programs, i.e., RAW register dependency distance and data/instruction reuse distance. The former serves to identify the degree of ILP in the program, while the latter captures the temporal and spatial locality behavior of the program. Unlike general classification of programs into processor bound and memory bound [9], this approach provides an opportunity to quantitatively analysis the relationship between the program characteristics and the program resource requirement implications. The following subsections give detailed description of the mapping heuristic with respect to these two aspects.

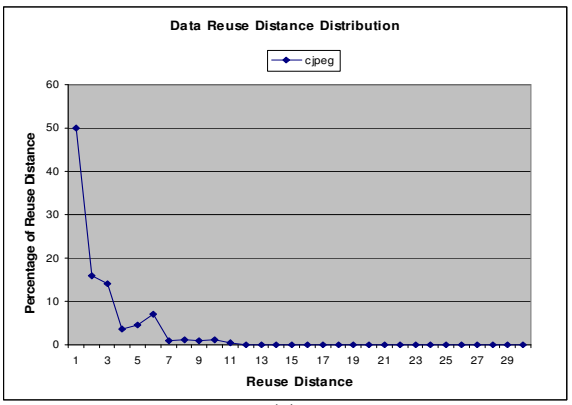
A. Correlation between Cache Size and Data/Inst Reuse Distance

According to the definition, reuse distance is measured by the number of memory accesses between two consecutive accesses to the same block address. In the worst case, these memory accesses are unique and only caches with cache block number larger than the reuse distance can possibly accommodate all the data set. Therefore, the percentage of reuse distance larger than the number of cache blocks in the cache is highly correlated with the cache miss rate of the program. Given the fixed cache block size, the cache capacity has to be enlarged to reduce the miss rate under this analytical model. However, the performance gain has to be large enough to amortize the increased hardware cost and energy consumption. Let C_i be the size of cache i, and B be the cache block size, define the switching gain when switching from one cache configuration to another as

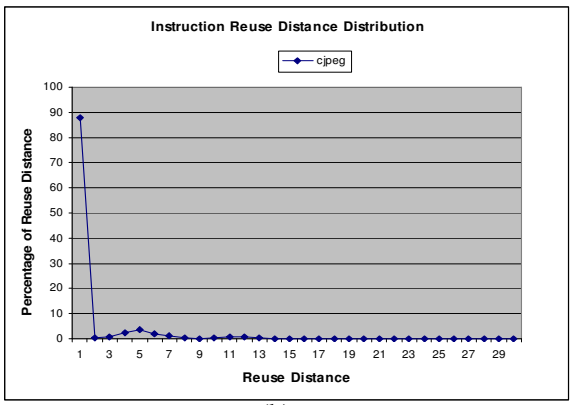
$$\left(\sum_{0\le i\le \log_2 \frac{C_j}{B}} P_i - \sum_{0\le i\le \log_2 \frac{C_k}{B}} P_i\right)/(C_j / C_k) \qquad (1)$$

where P_i and P_j are the percentages of reuse distance 2^i and 2^j respectively.

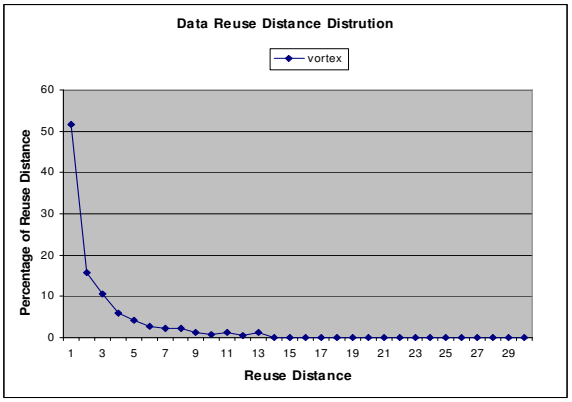
In this paper, data/instruction reuse distance distribution is profiled based on the cache block size consistent with the corresponding cache block size in the hypothetical heterogeneous multi-core system. Figure 2 shows the distribution of the data/instruction reuse distance, where x-axis represents the distance in logarithm of 2. According to the distribution, it is able to derive the switching gains


Data Reuse Distance Distribution
cjpeg
Percentage of Reuse Distance
Reuse Distance

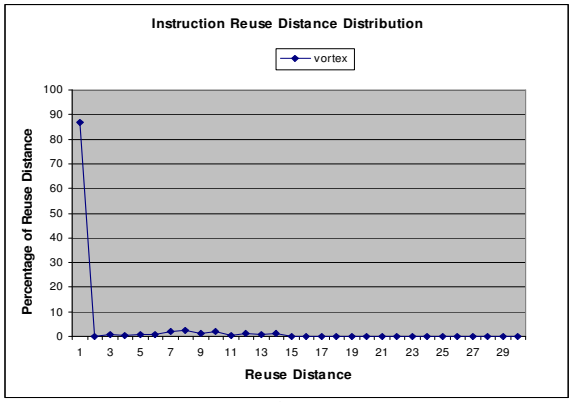
(a)


Instruction Reuse Distance Distribution
cjpeg
Percentage of Reuse Distance
Reuse Distance

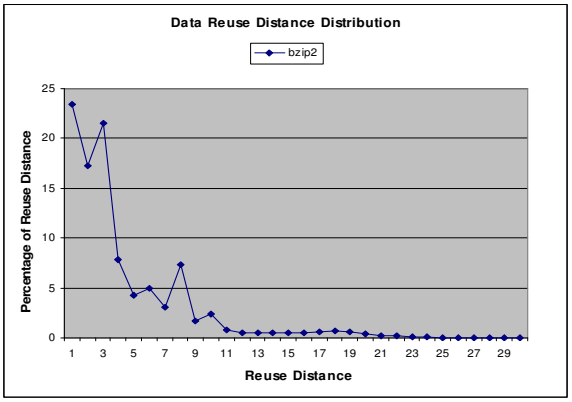
(b)


Data Reuse Distance Distrution
vortex
Percentage of Reuse Distance
Reuse Distance

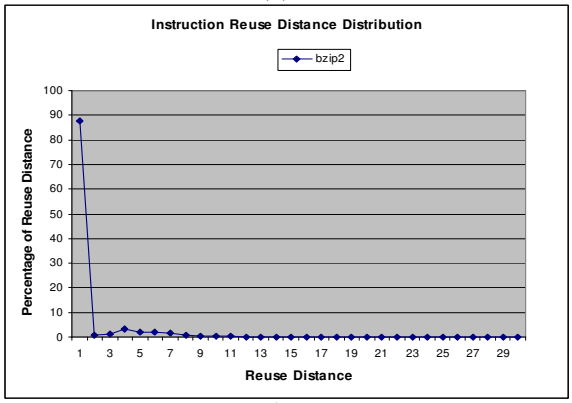
(c)


Instruction Reuse Distance Distribution
vortex
Percentage of Reuse Distance
Reuse Distance

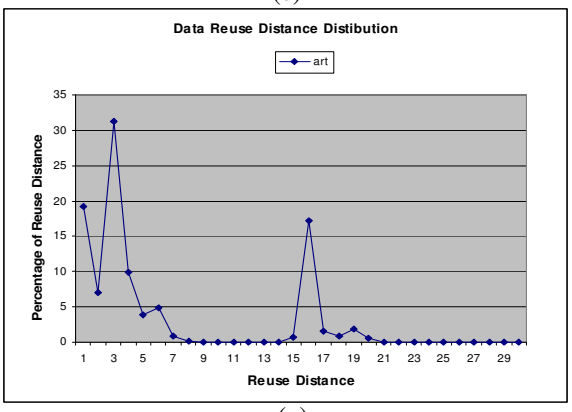
(d)


Data Reuse Distance Distribution
bzip2
Percentage of Reuse Distance
Reuse Distance

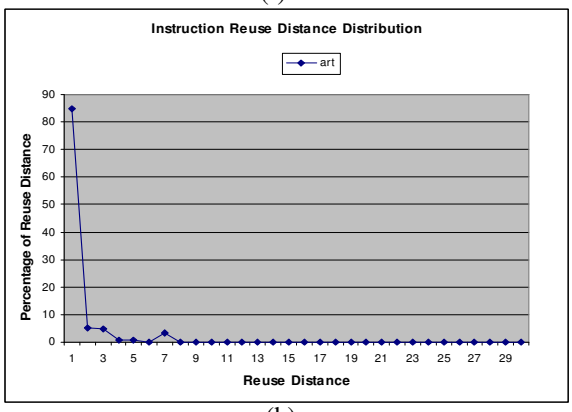
(e)


Instruction Reuse Distance Distribution
bzip2
Percentage of Reuse Distance
Reuse Distance

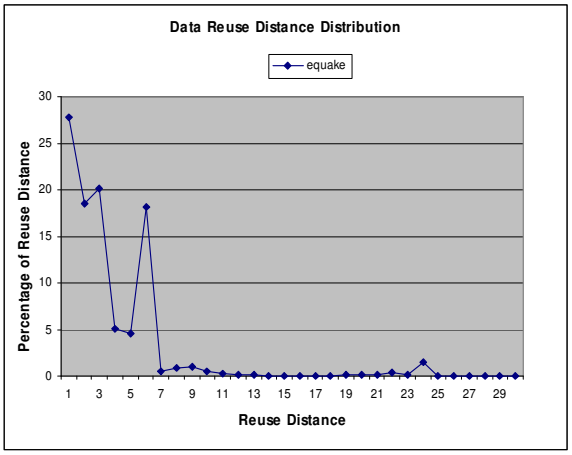
(f)


Data Reuse Distance Distibution
art
Percentage of Reuse Distance
Reuse Distance

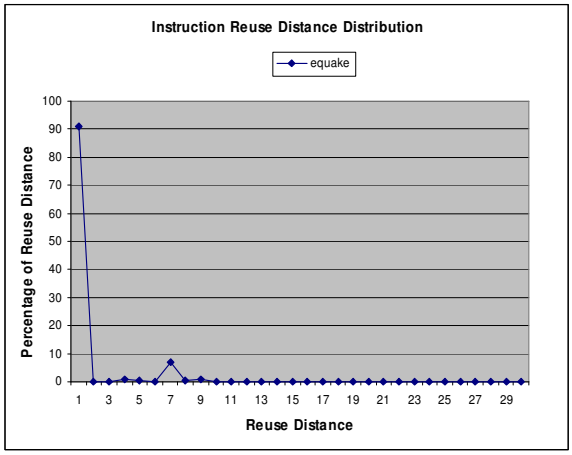
(g)


Instruction Reuse Distance Distribution
art
Percentage of Reuse Distance
Reuse Distance

(h)



(i)



(j)

Figure 4. Reuse distance distribution for five representative programs. (a),(c),(e),(g),(i). Data reuse distance distribution. (b),(d),(f),(h),(j). Instruction reuse distance distribution.

Table IV Cache Switching Gains with Respect to the Caches in In-order Core

Bench mark	Data Switching Gains			Instruction Switching Gains			Combined Switching Gains			Rank		
	Cfg2	Cfg3	Cfg4	Cfg2	Cfg3	Cfg4	Cfg2	Cfg3	Cfg4	Cfg2	Cfg3	Cfg4
Cjpeg	0.42	0.49	0.36	0.52	0.37	0.21	0.94	0.86	0.57	2	1	3
Vortex	1.10	1.08	0.69	1.01	1.07	0.71	2.11	2.15	1.4	2	1	3
Bzip2	1.56	2.61	1.52	0.85	0.59	0.36	2.41	3.2	1.88	2	1	3
Art	4.33	2.17	1.09	1.74	0.9	0.45	6.07	3.07	1.54	1	2	3
Equake	0.23	0.33	0.28	3.48	1.8	0.98	3.71	2.13	1.26	1	2	3

for both data and instruction caches by sweeping through four different configuration nodes. The efficiency of the overall L1 cache systems of the corresponding core can be represented by the sum of data and instruction cache switching gains. Table IV shows the switching gains of both instruction cache and data cache when the program is switched from the in-order core to other cores. The combined switching gain is the sum of switching gains on data and instruction caches of the corresponding configuration node. The higher the gain is, the more desirable the program should be mapped to the corresponding cache system. Such tendency can be captured by a rank system with 1 represents the highest priority and 3 the lowest priority.

B. Correlation between Issue Width and RAW Dependency Distance

RAW dependency distance sets an upper-bound of how much parallelism can be exploited in the program. In the best case, with the dependency distance *K*, *K-1* instructions can be executed simultaneously [8]. These instructions can exercise the processor with issue width less than *K-1* to its full potential. In other words, the product of the issue width with the percentage of instructions that has dependency distance larger than the issue width closely represents the maximum throughput when executing the program on the core. Therefore, similar gain based approach can be applied to determine the proper issue width. We define issue width switching gain as follows:

$$(W_i \bullet P_{dis>W_i} - W_j \bullet P_{dis>W_j}) / (W_i / W_j) \qquad (2)$$

where *Wi* and *Wj* stand for the issue width of the configuration node i and j, and $P_{dis>Wi}$ stands for the percentages that the reuse distance is larger than *Wi*.

Figure 5 shows RAW dependency distance distribution for both integer and floating-point programs, where *art* and *equake* are in floating-point dependency distance and the rest of the programs are in integer dependency distance. Although *cjpeg* has floating-point dependencies, they are

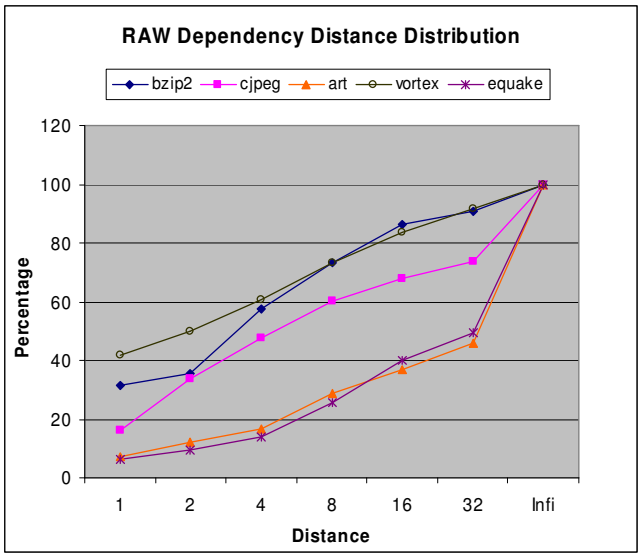


Figure 5. Distribution of RAW Dependency Distance for the representative programs.

Table V Issue Width Switching Gain with Respect to the Single-issue In-order Processor

Bench mark	Switching Gains			Rank		
	Cfg2	Cfg3	Cfg4	Cfg2	Cfg3	Cfg4
Cjpeg	24.47	31.40	29.28	3	1	2
Vortex	21.11	24.62	19.32	2	1	3
Bzip2	30.25	25.29	18.31	1	2	3
Art	41.4	60.33	59.51	3	1	2
Equake	43.87	62.46	62.71	3	2	1

in very small number in terms of the dynamic instruction count and have very little impact on the overall ILP for the program. Therefore, floating-point dependencies are ignored in *cjpeg*. The same is true for integer dependency part in *art* and *equake*. Applying equation (2) to the distribution, one can get the issue width switching gain of configuration 2 to 4 relative to the single-issue in-order core, as is shown in Table V. Rank system similar with the one used in ranking the cache switching gains is also employed here.

Now that we have two different rank systems that indicate two types of mapping preferences, the mapping heuristic should consider both and generate the application mapping result based on the following rules:

1. Choose the core that has smallest value in the sum of the rank pair unless there is a tie.
2. If there are ties, give priority to the rank on the issue width side. For example, if the rank pairs are (3,1),(1,3) and (2,2), the configurations with rank 1 in issue width should be chosen.

The second rule is based on the fact that out-of-order execution core can hide some memory latencies caused by L1 cache miss. The application-core mapping results can be derived according to these rules, and are shown in Table VI.

Table VI Mapping results of the programs

Bench mark	Rank Pairs			Mapping		
	Cfg2	Cfg3	Cfg4	Cfg2	Cfg3	Cfg4
Cjpeg	(2,3)	(1,1)	(3,2)		√	
Vortex	(2,2)	(1,1)	(3,3)		√	
Bzip2	(2,1)	(1,2)	(3,3)	√		
Art	(1,3)	(2,1)	(3,1)		√	
Equake	(1,3)	(2,2)	(3,1)			√

C. Evaluation of Mapping Result

In this paper, the quality of the mapping result is evaluated with energy-delay product. In other words, the application should be mapped to the core that has the lowest energy-delay product value after finishing executing the program. We use Wattch to get the average power number per cycle, and multiply it with the number of simulation cycles twice to calculate the effective energy-delay product, as is shown in table VII.

Table VII Normalized Energy Delay Product for benchmark programs across four different configurations

Benchmark	Cfg1	Cfg2	Cfg3	Cfg4	Best Cfg
Cjpeg	1	0.617	0.468	0.685	Cfg3
Vortex	1	0.534	0.440	0.554	Cfg3
Bzip2	1	0.618	0.685	0.862	Cfg2
Art	1	0.387	0.267	0.231	Cfg4
Equake	1	0.481	0.509	0.404	Cfg4

Table VII also shows the optimum mapping result for the corresponding program according to the minimum energy-delay product among four different configurations. Comparing the mapping result with the one derived by the proposed heuristic, four out of the five programs hit the optimum mapping. Even for art, which is the only one that has different mapping, the energy-delay product is very close to the optimum one. This gives a good confidence that by analyzing the micro-architecture independent characteristics, we are able to correlate program and its optimum core in a heterogeneous multi-core system. However, the proposed method has inherent deficiency because it cannot map any applications to the core with the in-order processor. The reason for that is the in-order core is

used as the baseline processor to calculate the switching gains. One way to get around this is to introduce a virtual baseline core configuration, and calculate the gains in terms of that core. Further investigation is required as to determine what the proper configuration of this virtual baseline processor is.

V. Conclusion & Future Work

This paper presents a method to map applications to its optimum cores in a heterogeneous multi-core system by analyzing the micro-architecture independent characteristics of these applications. The proposed method exploits data/instruction reuse distance and RAW dependency distance to derive the ranks for each configuration based on the switching gains, and maps the application to the core accordingly. The experiment result shows that four out of the five programs under study are mapped to the optimum core in terms of energy-delay product. While it is static mapping, this study opens the possibility to design a more intelligent dynamic program scheduling mechanism in heterogeneous multi-cores than the current trial-and-error approach.

However, several aspects of the proposed method require further research. The paper assumes that the proposed switching gain can correlate program characteristics with micro-architecture configurations. Systematic validation is required to understand how strong and general the correlation is. In addition, the heterogeneous multi-core system in this study assumes no memory sharing in L2 cache. Further research is required to identify the effect of L2 cache sharing on application mapping. Finally, we need to explore ways to apply the method to dynamic core switching.

References:

[1] Rakesh Kumar, Dean M. Tullsen, Norman P. Jouppi, "Core architecture optimization for heterogeneous chip multiprocessors" *Proceedings of the 15th international conference on Parallel architectures and compilation techniques, Sept. PACT '06.*

[2] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen. "Single-ISA Heterogeneous Multi-core Architectures: The Potential for Processor Power Reduction" *In International Symposium on Microarchitecture, Dec. 2003.*

[3] R. Kumar, D.M Tullsen, P. Ranganathan, N. P. Jouppi, and K.I. Farka, "Single-ISA Hetereogeneous Multi-core Architectures for Multithreaded Workload Performance". *In International Symposium on Computer Architecture, June 2004.*

[4] A. Phansalkar, A. Joshi, L. Eeckhout, and L. K. John, "Measuring Program Similarity: Experiments with SPEC CPU Benchmark Suites,". *IEEE International Symposium on Performance Analysis of Systems and Software. pp10-20. Mar.2005*

[5] SimpleScalar LLC, D. Burger and T. M. Austin. The simplescalar tool set version 3.02 http://www.simplescalar.com/

[6] Simpoint 3.0, Erez Perelman, Greg Hamerly and Brad Calder. "Picking Statistically Valid and Early Simulation Points", *In the International Conference on Parallel Architectures and Compilation Techniques, Sept. 2003.*

[7] Sim-Wattch 1.02, David Brooks, Vivek Tiwari, and Margaret Martonosi. "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," *27th International Symposium on Computer Architecture, June, 2000.*

[8] Pradeep K.Dubey, George B.Adams, and Michael J. Flynn, "Instruction Window Size Trade-offs and Characterization of Program Parallelism", *IEEE Trans. On Computers. Vol.43, No.4, April 1994.*

[9] Jaehyuk Huh, Stephen W. Keckler, Doug Bruger, " Exploring the Design Space for Future CMPs", *10th International conference on Parallel Architectures and Compilation Techniques, Sept. 2001.*