Energy-Aware Application Scheduling on a Heterogeneous Multi-core System

Jian Chen and Lizy K. John
The Electrical and Computer Engineering Department
The University of Texas at Austin
{jchen2, ljohn}@ece.utexas.edu

Abstract

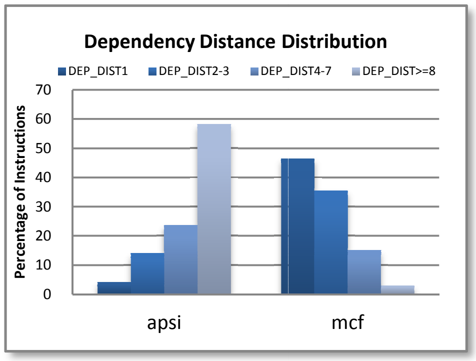
Heterogeneous multi-core processors are attractive for power efficient computing because of their ability to meet varied resource requirements of diverse applications in a workload. However, one of the challenges of using a heterogeneous multi-core processor is to schedule different programs in a workload to matching cores that can deliver the most efficient program execution. This paper presents an energy-aware scheduling mechanism that employs fuzzy logic to calculate the suitability between programs and cores by analyzing important inherent program characteristics such as instruction dependency distance and branch transition rate. The obtained suitability is then used to guide the program scheduling in the heterogeneous multi-core system. The experimental results show that the proposed suitability-guided program scheduling mechanism achieves up to 15.0% average reduction in energy-delay product compared with that of the random scheduling approach. To the best of our knowledge, this study is the first to apply fuzzy logic to schedule programs in heterogeneous multi-core systems.

I. Introduction

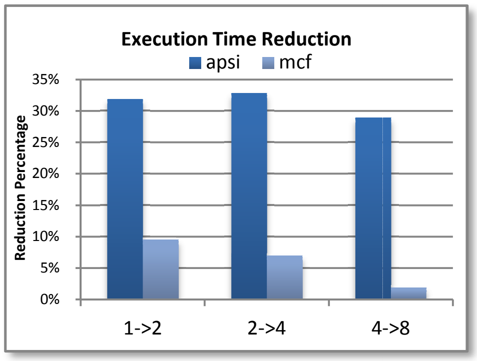
Employing heterogeneity in multi-core processor design is demonstrated to be an effective approach toward power efficient computing. By integrating different types of cores in a single chip, the heterogeneous multi-core processor provides the architectural capability to accommodate the diverse computational requirements of the applications. It achieves efficient computing by running the application on the core that is most suitable for its execution in terms of energy delay product (EDP)[2]. The recent endeavors in industry (CELL[11]) as well as in academia (Core Fusion [9], TFlex [8]) further underscore the importance of heterogeneity in the multi-core system.

While the heterogeneous multi-core provides the architectural support to match the programs' diverse resource requirements, it is the program scheduling mechanism that leverages this architecture opportunity to energy efficient computing. However, the problem of how to schedule the program to the desired core has not been well solved and remains an open question. Prior research on program scheduling in heterogeneous system mainly focuses on scheduling the subtasks of the applications in order to minimize the overall subtask execution time [12][13]. This execution time driven scheduling mechanism is no longer appropriate in modern heterogeneous multi-core processors since the power consumption, in addition to the performance, stands out as another major consideration in designing a good scheduling algorithm. More recently, Kumar et.al [2] proposed a program scheduling mechanism based on dynamic core selection to account for the energy efficiency of the program execution. It tentatively runs the application on neighboring or all cores, samples the performance and power characteristics of the application during these tentative runs, and chooses the desired core based on the sampled data. Although it could adapt to the program phase changes, this trial-and-error scheduling method has a significant energy overhead not only in the core context switching but also in the additional cache snooping after program migration [16]; therefore it does not achieve the potential the heterogeneous multi-core processor has to offer.

These existing scheduling methods did not exploit the relationship between a program's inherent characteristics and its hardware resource requirements. In fact, a program's hardware demands are governed by its inherent characteristics. For example, consider the instruction dependency distance distribution of a program (the number of instructions between the producer of a data and its consumer). As shown in Figure 1(a), the SPEC benchmark *apsi* has a large percentage of instructions with long dependency distance, while *mcf* has a high percentage of instructions with short dependency distance. These two opposite trends in dependency distance distribution indicate different amounts of instruction level parallelism (ILP) in these two programs, and hence different requirements of instruction issue width on the processor core. As shown in Figure 1(b), *apsi* demonstrates a near constant reduction rate in execution time as the instruction issue width goes from 1 to 8. This is because the program has sufficient ILP, as indicated in the dependency distance distribution, to keep up with the issue width scaling, and hence favors processor core with large issue width. On the other hand, *mcf* has a significantly lower reduction rate in execution time, and the amplitude of the rate sharply goes



(a)



(b)

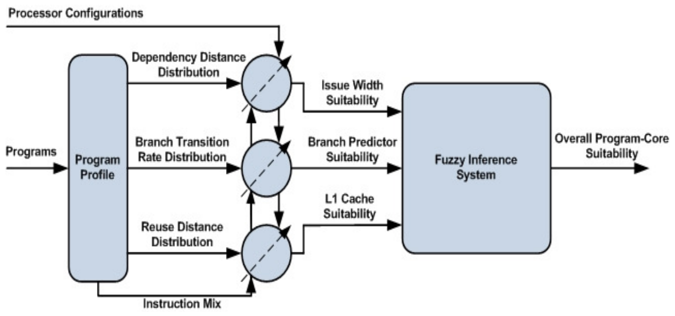
Figure 1. Instruction dependency distance distribution and execution time reduction of *apsi* and *mcf*.

down as the instruction issue width gets larger, which means the program is more suitable to run on a core with small issue width. Therefore, programs' inherent characteristics shape its hardware resource demands, and could be used to guide the program scheduling in heterogeneous multi-cores.

This paper presents a suitability-based approach to leverage this intuitive, yet fuzzy, relationship between programs and cores for the program scheduling in heterogeneous multi-core processors. As shown in Figure 2, the proposed method profiles the applications to obtain important micro-architecture independent program characteristics [3], such as the dependency distance distribution, the reuse distance distribution and the branch transition rate distribution. These characteristics determine the ILP, the data locality as well as the branch predictability of the applications, which largely define the applications' overall resource demands. The profiled characteristics, coupled with the corresponding hardware configurations, are used to generate the suitability degree for issue width, cache size and branch predictor size respectively. These degrees go through a fuzzy inference system, where we can integrate human knowledge in its rule system, to produce an overall suitability degree that represents the degree of the match between the program and the core. We demonstrate that the overall suitability degree has a strong enough correlation with EDP to generate high quality program scheduling in heterogeneous multi-cores. The proposed scheduling method provides up to 15.0% EDP reduction compared with the average EDP of random scheduling. The contributions of the paper include:

- Presents a model to measure the suitability between each characteristic and its corresponding hardware configuration.
- Employs fuzzy logic in determining the overall suitability to guide the program scheduling for efficient computing in heterogeneous multi-cores.

The rest of the paper is organized as follows: Section II presents the micro-architecture independent characteristics investigated in this paper as well as the suitability metrics for these characteristics. Section III describes the fundamentals and the implementation of the fuzzy inference system employed in this paper. Section IV gives the setup of the experimental environment. Section V discusses the experimental results. Section VI summaries the related work, and Section VII concludes the paper.



Processor Configurations
Programs
Program Profile
Dependency Distance Distribution
Branch Transition Rate Distribution
Reuse Distance Distribution
Instruction Mix
Issue Width Suitability
Branch Predictor Suitability
L1 Cache Suitability
Fuzzy Inference System
Overall Program-Core Suitability

Figure 2. The overall flow to generate the program-core suitability. The instruction mix is used to tune the suitability degrees of the three characteristics.

II. Program Characteristics and Suitability Metrics

The proposed method exploits three major inherent characteristics of the programs. Each characteristic is associated with a suitability metric, which measures the degree of the match between that characteristic and the corresponding hardware configuration. This section describes these inherent program characteristics as well as the models to obtain the corresponding suitability metrics.

A. Instruction Dependency Distance and Issue Width Suitability

The issue width suitability attempts to measure the match between the program's ILP and the processor's issue width. The program's ILP could be captured with instruction dependency distance, which is defined as the total number of instructions in the dynamic instruction stream between the producer and the *first* consumer of a register instance [3]. Unlike the conventional read-after-write (RAW) dependency distance, the instruction dependency distance followed by this definition excludes the non-critical RAW dependencies, hence is more accurate in representing the program's ILP. Specifically, for a given dependency distance distribution, the higher the percentage of instructions with long dependency distance is, the larger the amount of ILP in the program would be.

To calculate the issue width suitability, we classify the distances into four groups according to the dependency distance distribution, i.e., group 1 with distance of 1, group 2 with distance of 2-3, group 3 with distance of 4-7, and group 4 with distance of 8 and larger. Each group has its most suitable issue width to exploit its parallelism, that is, issue width of 1 for group 1, issue width of 2 for group 2, issue width of 4 for group 3, and issue width of 8 for group 4. Let X_i, $i=1..4$, represent the issue width from 1 to 8, then the mass center (or the weighted average) of the distribution would be $\sum_{i=1}^{4} P_i * X_i \,/\, \sum_{i=1}^{4} P_i$,where Pi is the percentage of instructions whose dependency distance falls in group i. This mass center of the distribution indicates where the application locates in the spectrum of issue widths. Therefore, the distance between the center of mass and the node representing certain issue width shows how close the program's ILP matches the core's issue width, and hence could be the degree of issue width suitability when mapping the application to that particular core. The equation is shown as follows:

$$IssueWidthSuitability(i) = \left| X_i - \sum_{i=1}^{4} P_i * X_i \Big/ \sum_{i=1}^{4} P_i \right| \qquad (1)$$

This degree, however, has the opposite meaning with the original definition of the suitability since the smaller the degree is, the closer the distance is, and hence the higher the suitability would be. Nevertheless, this degree can still be applied in our fuzzy inference system by complementing the corresponding conditions in the fuzzy rule base.

B. Branch Transition Rate and Branch Predictor Suitability

The branch predictor suitability tries to measure the match between program's branch predictability and the branch predictor size. To capture the branch predictability of the program, we use branch transition rate, which is demonstrated to be an appropriate metric for the branch predictability of the program by Huang et.al [14]. Generally speaking, the branch instructions with extremely low and extremely high transition rate are easy to predict since the branch history pattern of these instructions could be captured with short history registers. As the transition rate approaches 50%, it becomes harder to predict the branch results since longer history register is required to capture the history pattern of these branch instructions. Based on this observation, we evenly divide the transition rates into 10 buckets. Specifically, we have the buckets [0, 0.1], [0.1, 0.2], [0.2, 0.3] … [0.9, 1.0]. Each bucket i has its percentage Pi representing the amount of branch instructions whose transition rate falls in that range. Since the branch instructions in the buckets [0.4, 0.5] and [0.5, 0.6] are the hardest to predict, they are associated with the largest branch predictor. The branch instructions in the buckets [0.3, 0.4] and [0.6, 0.7] are relatively easier to predict, and hence are associated with a smaller branch predictor. Same trend applies in buckets [0.2, 0.3] and [0.7, 0.8], and buckets [0.1, 0.2] and [0.8, 0.9]. Therefore, similar with the way of calculating issue width suitability, we have the following equation to calculate the branch suitability:

$$BranchSuitability(i) =$$

$$\left| B_i - \frac{(B_1*(P_2+P_9)+B_2*(P_3+P_8)+B_3*(P_4+P_7)+B_4*w*\sum_{i=5}^{6} P_i)}{\sum_{i=2}^{4} P_i+\sum_{i=7}^{9} P_i+w*\sum_{i=5}^{6} P_i} \right| \qquad (2)$$

where B_i, i=1..4, are the x coordinates of the nodes representing the sizes of the branch predictors, organized in an increasing order with B_1 the smallest and B_4 the largest (This study only considers four different-sized branch predictors of the same type). We do not consider the buckets [0, 0.1] and [0.9, 1] because branch instructions in this range are very easy to predict, and even the smallest branch predictor in this study would be more than enough for them. The parameter w is used to tune the weight of the largest branch predictor, and equals $\alpha \times P_{cond}$. α is an empirically determined value, and increases as the instruction issue width increases. It is used to keep track of the fact that as the issue width gets wider the branch misprediction penalty also increases, and hence a larger branch predictor with higher prediction accuracy is more desirable. P_{cond} is the percentage of the conditional branches in the instruction mix. A large P_{cond} leads to a large number of hard-to-predict branches, and hence the weight of large branch predictor should be high.

Like the issue width suitability, Equation (2) calculates the distance between the mass center of the transition rate distribution and the node representing certain branch predictor size. Again, this distance has the opposite meaning with the original definition of the suitability. Therefore, the corresponding conditions in the fuzzy rule base need to be complemented before using this distance as the suitability for the branch predictor.

C. Data Reuse Distance and L1 Data Cache Suitability

The cache suitability attempts to measure the degree of the match between the program's data locality and the cache size. The program's data locality is characterized with data reuse distance, which is defined as the number of unique memory accesses between two consecutive memory accesses to the same block address[15]. By grouping the data accesses in terms of their reuse distances, we can have the reuse distance distribution, which gives the percentages of the data accesses with a certain reuse distance among the total data accesses. However, due to the lack of well-defined relationship between the reuse distance and the corresponding desired L1 cache size, the idea of using the center of mass to represent the figure of merit could not be directly applied to reuse distance distribution. Therefore, we introduce a different metric, *cache efficiency*, to measure the suitability between the locality of the program and the L1 cache size. The cache efficiency is defined as $P_{R<C}/C$, where C is the L1 data cache size and $P_{R<C}$ is the percentage of the data accesses with reuse distance less than C. The cache efficiency essentially calculates how much

program locality per unit cache size captures. Since the value of suitability has to be in the range between 0 and 1, the cache efficiency should be normalized before it can be used as the cache suitability, which is shown in the following equation:

$$CacheSuitability(i) = \frac{P_{R<Ci}/C_i}{(P_{R<C}/C)_{max}} \quad (3)$$

where C_i is the L1 data cache size of core i in the heterogeneous multi-core processor, and $(P_{R<C}/C)_{\max}$ is the largest cache efficiency a program can have when it is mapped to the cores with different L1 data cache sizes.

III. Fuzzy Inference System

This paper employs fuzzy logic to combine individual suitability metrics and produce an overall suitability that indicates the overall degree of the match between a program and a core. Fuzzy logic allows explicit human knowledge representation using linguistic "IF-THEN" rules, and thus is more applicable in the situations like the matching between programs and cores, where well-defined deterministic mathematical model is not available. In this section, we first briefly introduce the fundamentals of fuzzy logic, and then describe the design of the fuzzy inference system for the program-core suitability.

A. Fuzzy Inference System (FIS)

Unlike Boolean logic, the fuzzy inference system uses a collection of membership functions and the built-in linguistic rules to map the inputs to an output. It is mainly composed of four steps: fuzzification, inference, composition, and defuzzification [10], as shown in Figure 3.

The fuzzification process transforms the crisp input values into fuzzy degrees via input membership function evaluation. This step is necessary because the rules representing human knowledge is reasoned with the fuzzy sets. Then in the inference step, the fuzzy operator (AND or OR) is applied to two or more fuzzified input variables to obtain a number that represents the result of the premise for the rule. This number is used to truncate the corresponding output fuzzy set that represents the output of the rule. The truncated fuzzy sets are aggregated into a single fuzzy set during the composition step. Finally, the defuzzification process converts this single fuzzy set back to a crispy value, usually by calculating the center of area under the curve.

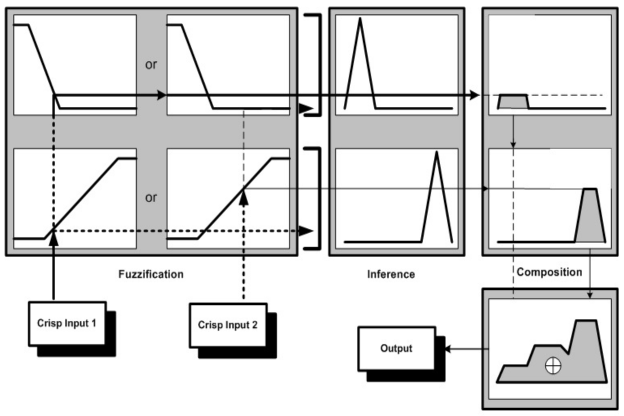


Figure 3. Fuzzy inference system

B. Implementation of FIS

The task of FIS is to convert three suitability degrees, namely, the issue width suitability, the cache suitability and the branch predictor suitability, to an overall suitability for the program-core pair. Each suitability degree is assigned with two possible fuzzy sets, namely, "low" and "high", leading to total eight output membership functions. As shown in Figure 4, the input membership functions are designed in such a way that both "low" and "high" are triangle-shaped with "low" covering any value between 0 and 0.9 and "high" covering any value between 0.1 than 1.0. In addition, each output membership function is associated with a name, which is used in the rules to represent the linguistic meaning of the suitability level.

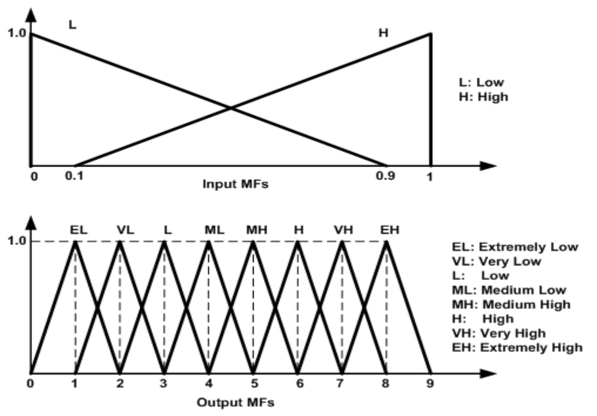


Figure 4. Input and output membership functions. The output MFs divide the range [0,1] into 8 equal segments.

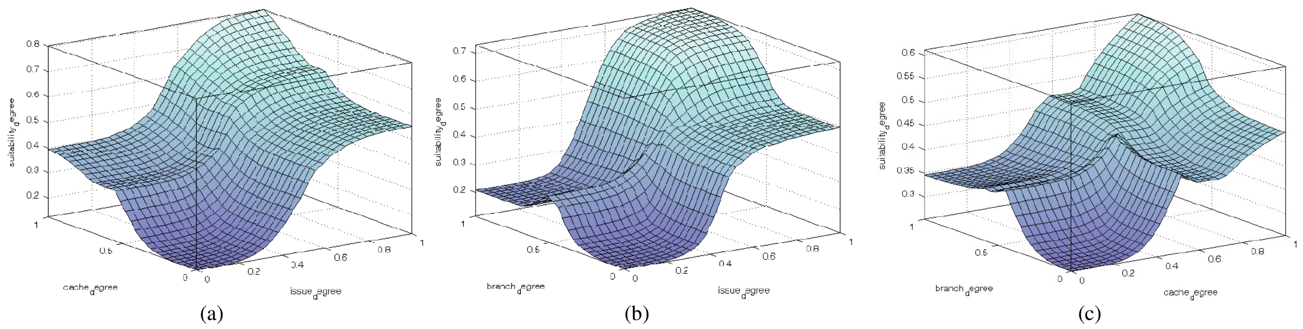


Figure 5. Response surface of the fuzzy inference system

Table I lists the rules employed in the fuzzy inference system. Each rule establishes the mapping relationship between the three individual suitability degrees and the overall suitability degree based on human knowledge about this relationship. For example, if all individual suitability degrees are low, which means the program doesn't fit the core from any of the three aspects, the overall suitability is lowest, or extremely low (EL). Similarly, if all individual suitability degrees are high, the overall suitability is highest, or extremely high (EH). The rest of the rules are designed such that the three fuzzy variables in the IF conditions have different priorities, with "issue width suitability" the highest, and the "branch predictor suitability" the lowest. This is the case because instruction issue width usually has the most significant impact on the execution efficiency of the program, followed by the L1 cache size and the branch predictor if we do not consider L2 cache. Figure 5 shows the response surface of these rules. Three subfigures are required to display the overall response surface of the three-input fuzzy inference system. Each subfigure has two input variables, with the third input set to zero.

Table I. Rules of the Fuzzy Inference System

IF	**THEN**
(issue width suitability is low) **AND** (cache suitability is low) **AND** (branch predictor suitability is low)	(overall suitability is EL)
(issue width suitability is low) **AND** (cache suitability is low) **AND** (branch predictor suitability is high)	(overall suitability is VL)
(issue width suitability is low) **AND** (cache suitability is high) **AND** (branch predictor suitability is low)	(overall suitability is L)
(issue width suitability is high) **AND** (cache suitability is low) **AND** (branch predictor suitability is low)	(overall suitability is ML)
(issue width suitability is low) **AND** (cache suitability is high) **AND** (branch predictor suitability is high)	(overall suitability is MH)
(issue width suitability is high) **AND** (cache suitability is low) **AND** (branch predictor suitability is high)	(overall suitability is H)
(issue width suitability is high) **AND** (cache suitability is high) **AND** (branch predictor suitability is low)	(overall suitability is VH)
(issue width suitability is high) **AND** (cache suitability is high) **AND** (branch predictor suitability is high)	(overall suitability is EH)

IV. EXPERIMENT SETUP

We created a hypothetical single-ISA heterogeneous multi-core processor with four different cores. The configurations of these cores shall demonstrate enough heterogeneity so that the mapping of an application to different cores could yield noticeable difference in terms of performance and energy consumption. Although the types of the cores could be ranging from ASIC accelerator to VLIW processor, this paper only focuses on out-of-order superscalar processor cores with variations in instruction issue width, L1 data cache size and branch predictor size.

Table II. Configuration Options for Three Key Parameters

Items	**Configuration Options**
Issue Width	single-issue, 2-issue, 4-issue, 8-issue
L1 D-Cache	16KB, 4-way, block size 64byte, 32KB, 4-way, block size 64byte, 64KB, 4-way, block size 64byte, 128KB, 4-way, block size 64byte
Branch Predictor	1K Gshare, 2K Gshare, 4K Gshare, 8K Gshare

Table II gives the configuration options of these three parameters of the processor. To be consistent with the assumptions made in the previous section, each parameter has 4 possible options, leading to 48 possible core configurations. We assign both *Xi* and *Bi* (*i=1..4*), which are the x coordinates of the nodes representing the issue widths and branch predictor sizes, to be 0.125, 0.25, 0.5, and 1 so that the suitability degree would be in the range of [0,1]. These configurations are used to evaluate the effectiveness of the suitability metrics we proposed. We also compose our heterogeneous quad-core processor based on table II to evaluate the effectiveness of the proposed suitability-guided program scheduling. The detailed configurations of these cores are listed in Table III. Each core has a private 512K L2 cache with a hit latency of 12 cycles, and a miss latency of 100 cycles. The other parameters, including the load/store queue size, and the number of ALUs, are chosen in a way that the design of the core is balanced. We assume there is no resource sharing between the cores on the chip, and the communication and synchronization between the cores are not considered in this study.

Table III. Core Configurations for Multi-core Processor

Items	**Configurations**
Core 1	Out-of-order, 2-issue, Gshare(1k), 16k 4-way d-cache 64byte, 32k 2-way i-cache 64byte, 512k L2 cache
Core 2	Out-of-order, 2-issue, Gshare(2k), 32k 4-way d-cache 64byte, 32k 2-way i-cache 64byte, 512k L2 cache
Core 3	Out-of-order, 4-issue, Ghsare(4k), 32k 4-way d-cache 64byte, 32k 2-way i-cache 64byte, 512k L2 cache
Core 4	Out-of-order, 8-issue, Gshare(8k), 64k 4-way d-cache 64byte, 32k 2-way i-cache 64byte, 512k L2 cache

The application space of the experiment is composed of benchmark programs from SPEC CPU2000, with both integer and floating point benchmarks compiled to Alpha-ISA. We modified SimProfile from Simplescalar tool set [5] to instrument programs and collect the aforementioned characteristics. To reduce the time for profiling and simulation, each SPEC2000 program is profiled at its single

Simpoint interval with 100 million instructions [6] instead of the entire run of the program. Each Simpoint interval is simulated on Wattch [7] to collect the performance and power data. Since we assume there is no sharing and communication between the programs running on different cores, the overall EDP of the multi-core system is the sum of EDP of each core.

V. Experimental Results

To demonstrate the effectiveness of the suitability-based program scheduling in heterogeneous multi-core, we need to evaluate the quality of the individual suitability metric as well as the effectiveness of the overall suitability in guiding the program scheduling. This section presents the experimental results for these two.

A. Evaluation of Individual Suitability

To evaluate the instruction issue width suitability, we chose 4 cores with different instruction issue widths, but the rest of the hardware configurations are the same. We calculated the issue width suitability according to the equation (1) and ranked the cores according to the calculated suitability for each benchmark program. On the other hand, we also performed simulations of each program on these cores to obtain the corresponding EDPs, and ranked the cores according to the simulated EDPs. We use Spearman's rank correlation coefficient [17] to measure the quality of the issue width suitability metric, that is, how close the rank according to issue width suitability matches the rank according to the simulated EDP. Figure 6 shows the rank correlation coefficient of each benchmark program. 9 out of the 20 benchmark programs have the correlation coefficient of 1, which indicates a perfect match. The smallest observed rank correlation coefficient is 0.6. It should be noted that with 4 cores, the worst case rank correlation coefficient is -1. Therefore, the issue width suitability captures the match between the program's ILP and the issue width with a high accuracy.

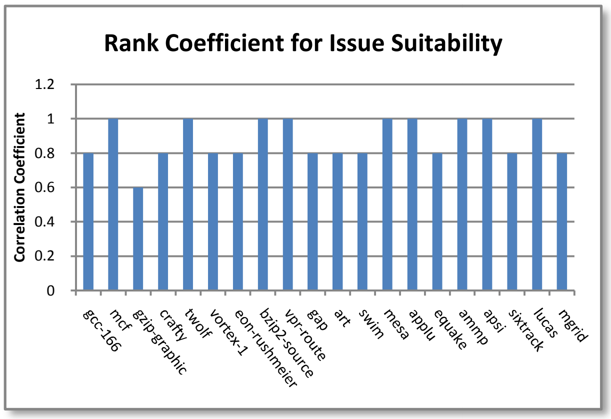


Figure 6. Rank correlation coefficient between the rank according to issue width suitability and the rank according to simulated EDP.

Similarly, to evaluate the branch predictor suitability, we selected three groups of processor cores. Each group is a collection of 4 cores with different branch predictors (as shown in table II), and the rest of the hardware configurations are the same across these 4 cores. The cores from different group have different instruction issue widths, ranging from 2 to 8. We calculated the branch predictor suitability according to the equation (2) and ranked the cores within each group according to the calculated suitability for each benchmark program. We then calculated the correlation coefficients between these ranks and the ranks according to the simulated EDP within each group. As shown in Figure 7, the smallest observed correlation coefficient is 0.2 (sixtrack), and the most of the other programs have a coefficient above 0.8. Also, as the issue width increases, the correlation coefficient increases or remains the same for most benchmark programs, which demonstrates the effectiveness of the weight we introduced in the equation (2). However, the programs, like *art* and *applu*, exhibit an opposite trend with the other ones. This is because these programs have a small optimum branch history length, and as the history length becomes larger, the branch predictor becomes less accurate, which is opposite with the assumption we made in the branch predictor suitability model. Nevertheless, the branch predictor suitability provides a good measurement of the match between the program's branch predictability and the branch predictor size.

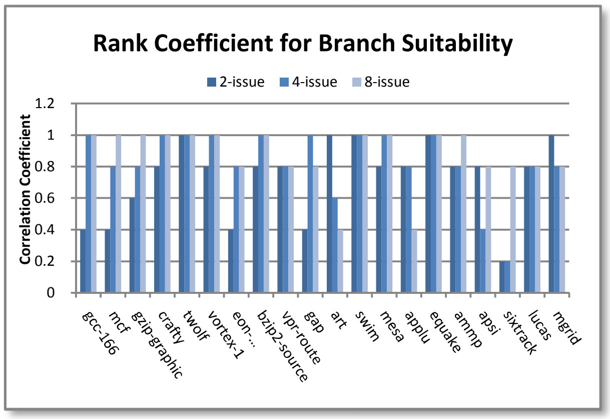


Figure 7. Rank correlation coefficient between the rank according to branch predictor suitability and the rank according to simulated EDP

To evaluate the cache suitability, we choose 4 cores with different L1 data cache sizes, and employ the method same as what we used in evaluating the suitability of instruction issue width to obtain the rank correlation coefficient. We observe that the rank correlation coefficient is 1.0 for each benchmark program, which means the proposed cache suitability perfectly captures the efficiency of L1 data cache. This is not surprising because each time L1 cache size doubles, the power consumption almost doubles yet the performance gain is much less. This effect of diminishing return is properly captured in equation (3).

B. Evaluation of the Overall Suitability

The objective of the overall suitability is to guide the program scheduling in the heterogeneous multi-core processor so that the total energy delay product would be minimized. In order to evaluate the quality of the overall suitability, we ran each program on each of the 48 possible cores exhaustively and ranked the cores in terms of the simulated EDPs. We also ranked these cores according to the overall suitability of each program-core pair. Figure 8 shows the rank correlation coefficient of the two sets of ranks. The minimum observed coefficient is 0.58, and the average coefficient is 0.81. This result shows the overall suitability captures the match between programs and cores with a high quality.

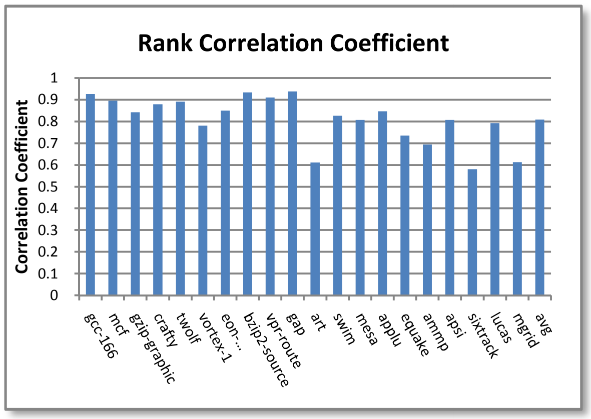


Figure 8. Rank correlation coefficient between the ranks according to the overall suitability and the ranks according to the overall simulated EDP

To evaluate the effectiveness of the overall suitability in program scheduling, we randomly chose 3 to 8 programs from the benchmark suite, and schedule them to the cores in our hypothetical heterogeneous quad-core processor, both randomly and under the guidance of the suitability. The random scheduling method schedules the program from the top of the queue to core 1, followed by core 2, core 3, and so on. The randomness is achieved by permuting the sequence of the programs in the program queue. The suitability-guided scheduling method first sorts the program-core pairs in terms of their overall suitability, and then selects the program with the highest rank for the given available core. In case there is a tie, the program in the leading position of the program queue takes precedence. We then compared the average overall EDP with suitability-guided scheduling against the average EDP with the random scheduling. Figure 9 shows the average EDP reduction achieved by suitability-guided scheduling. Note that the performance of suitability-guided scheduling continuously improves as the number of programs in the queue increase, with the average EDP reduction rate from 8.1% when the program number is 3 to 15.0% when the program number is 8. This is because as the number of programs to be scheduled increases, it is more likely for the suitability-guided scheduler to find the most suitable program for the available core, hence reduce the overall EDP cost. Also shown in Figure 8 is the average EDP reduction achieved by oracle scheduling. The oracle scheduling assumes that the EDP of each program-core pair is known even before the program has been executed on the core. The mechanism of the oracle scheduling is the same as that of the suitability-guided scheduling except that the oracle scheduling uses the EDP instead of the suitability to determine which program in the queue should be scheduled to the available core. The oracle scheduling is an ideal case, and it sets an upper bound of what different scheduling heuristics could achieve in average EDP reduction. Note that the average EDP reduction rate of the oracle scheduling increases from 9.0% to 21.6% as the number of the programs in the queue increases from 3 to 8. The maximum observed gap between the average EDP reduction of the suitability-guided scheduling and that of the oracle scheduling is 6.6%, which indicates the good quality of the suitability-guided scheduling.

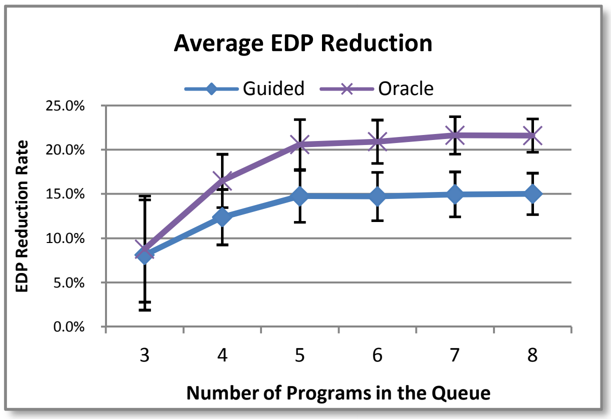


Figure 9. Average EDP reduction rates achieved with the suitability guided program scheduling and with the oracle scheduling when compared with that of the random scheduling. The error bars are the 95% confidence interval of the EDP reduction rate.

We also compared our suitability-guided scheduling with the trial-and-error scheduling proposed by Kumar [2]. To implement the trial-and-error scheduling algorithm, we tentatively ran the program on the cores for the first 2 million instructions and sampled the EDP during this interval. These sampled EDPs are used to guide the program scheduling such that the program with the minimum sampled EDP in the program queue is scheduled to the available core. When we calculated the overall EDP, we included the energy and execution time overhead caused by the cache cold start effect during the tentative runs. However, we did not consider the additional overhead caused by the context switching. Figure 10 shows the comparison of EDPs between these two scheduling mechanisms for several program combinations. As can be seen from the figure, the EDP of the suitability-guided scheduling is always less than that of the trial-and-error scheduling. Obviously the EDP overhead during the tentative runs significantly degrades the performance of the trial-and-error scheduling. In addition, the sampled EDP during the tentative runs may not accurately represent overall EDP of the entire program phase due to the interference of cold start effects. As a result, the scheduling based on the sampled EDP could possibly yield a large overall EDP of a certain program combination. Moreover, if we consider the context switching overhead in the tentative runs, the performance of trial-and-error scheduling could be even worse. Therefore, our suitability-guided scheduling mechanism offers an attractive alternative to the dynamic trial-and-error scheduling.

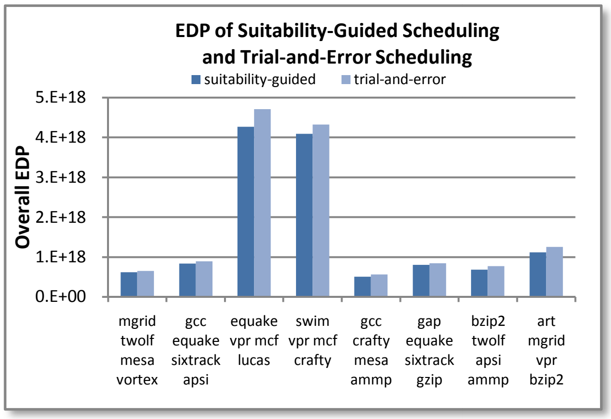


Figure 10. EDP comparison between suitability-guided scheduling and trial-and-error scheduling. Each program combination has four programs randomly chosen from the benchmark suite.

VI. Related Work

There have been several attempts to optimize the program scheduling in the heterogeneous computing environment. Siegel presented static and hybrid heuristic to schedule the subtasks in heterogeneous systems [12][13]. These methods were based on the accurate estimation of the execution time of each subtask with the objective of minimizing the overall completion time of the program. Our method, however, performs the scheduling on the granularity of entire program, and attempts to achieve efficient computing by minimizing the EDP of the heterogeneous multi-core processor.

Kumar et al.[1][2]discussed a dynamic scheduling approach based on sampling the program's behavior during the switching intervals. This method tentatively runs the program on different cores, each for a short period of time, and then schedules the program to the optimum core according to the sampled data during the tentative runs. The downside of this method is the expensive context switching cost of the tentative runs, which may significantly degrade the overall efficiency of the multi-core system. In addition, this trial-and-error approach does not scale well as the number of cores increases. In future many-core chips, sampling a large amount of cores before scheduling the program would be impractical because the extra cost of sampling will exceed the potential gain of core switching. Our method is static, therefore, there is no requirement for tentative runs, and hence no additional power overhead at runtime. In addition, our method is scalable since it is free of tentative runs.

Chen et.al [4] also did static application mapping in heterogeneous multi-cores based on micro-architecture independent characteristics. Their work is based on the switching gain, which requires one processor core as the reference. Our work is based on suitability and does not require any core on the chip to be as the reference.

VII. Conclusion

This paper presents a fuzzy logic based approach to schedule the program to its optimum core by analyzing key program characteristics such as the instruction dependency distance, the data reuse distance, and the branch transition rate. With the built-in human intelligence in its rule system, the proposed fuzzy logic method can measure the suitability of the hard-to-model program-core relationship and use that suitability to guide the program scheduling. The experiment results show that the proposed energy-aware scheduling method achieves up to 15% average reduction in energy-delay product compared with that of the random scheduling approach. The proposed method provides an attractive way to achieve stable and low, if not minimal, energy-delay product in the heterogeneous multi-core processor.

The future work of this research includes: employing more program characteristics in determining the suitability; considering the effects of resource sharing and inter-core communication; and extending the core type from out-of-order superscalar to other types such as VLIW processors and SIMD accelerators.

Acknowledgements

The authors would like to thank Dimitris Kaseridis, Ciji Isen, and the anonymous reviewers for their valuable feedback. This work is supported in part through the NSF award numbers 0702694. Any opinions, findings, and conclusions or recommendations expressed herein are those of the authors and do not necessarily reflect the views of the NSF.

References

[1] R. 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. 2006.*

[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] 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*

[4] Jian Chen, Nidhi Nayyar, and Lizy K. John, "Mapping of Applications to Heterogeneous Multi-cores Based on Micro-architecture Independent Characteristics", *Third Workshop on Unique Chips and Systems, ISPASS2007. April 2007.*

[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]. Kim, Changkyu; Sethumadhavan, Simha; Govindan, M.S.; Ranganathan, Nitya; Gulati, Divya; Burger, Doug; Keckler, Stephen W., "Composable Lightweight Processors," *40th Annual IEEE/ACM International Symposium on Microarchitecture, pp.381-394, 1-5 Dec. 2007*

[9]. E. İpek, M. Kırman, N. Kırman, and J.F. Martínez. “Core Fusion: Accommodating software diversity in chip multiprocessors”. *In Intl. Symposium. on Computer Architecture, San Diego, CA, June 2007*

[10]. Heintz-Jürgen Zimmermann, “Fuzzy Sets, Decision Making, and Expert Systems”. *Kluwer Academic Publishers, 1987.*

[11]. Hofstee, H.P., "Power efficient processor architecture and the cell processor," *11th International Symposium on High-Performance Computer Architecture, HPCA-11. pp. 258-262, Feb. 2005*

[12]. H.J. Siegel, Wang Lee, and V.P Roychowdhury, etc. “Computing with heterogeneous parallel machines: advantages and challenges”, *Proceedings. Second International Symposium on Parallel Architectures, Algorithms, and Networks, 12-14 June 1996.*

[13]. M. Maheswaran and H.J.Siegel, “A Dynamic Matching and Scheduling Algorithm for Heterogeneous Computing Systems”, *Proc. Heterogeneous Computing Workshop, pp. 57-69, 1998.*

[14]. M.Haungs, P.Sallee, M. Farrens, "Branch transition rate: a new metric for improved branch classification analysis," Proceedings. *Sixth International Symposium on High-Performance Computer Architecture. HPCA-6., pp.241-250, 2000*

[15]. T. Lafage and A. Seznec, “Choosing Representative Slices of Program Execution for Microarchitecture Simulations: A Preliminary Application to the Data Stream”, *Workshop on Workload Characterization (WWC-2000), Sept 2000*

[16]. C. S. Ballapuram, A. Sharif and Hsien-Hsin S. Lee, “Exploiting Access Semantics and Program Behavior to Reduce Snoop Power in Chip Multiprocessors”, *Proceedings of 13th International Conference on Architectural Support for Programming Languages and Operating Systems, pp 60-69, March 2008.*

[17]. M. Hollander, D.A. Wolfe, "Nonparametric statistical methods", *Wiley 1973.*