A Neuroevolution Method for Dynamic Resource Allocation on a Chip Multiprocessor [1]

Faustino J. Gomez, Doug Burger, and Risto Miikkulainen
The University of Texas
Department of Computer Sciences
Austin, TX 78712
(inaki,dburger,risto@cs.utexas.edu)

Abstract

Technology-driven limitations will soon force microprocessor chips to contain multiple processing cores, as the scalability of individual cores peaks but transistor counts continue to increase. To obtain best performance, flexible management of the on-chip resources, such as cache memory and off-chip bandwidth, is needed. However, control for the dynamic management of these on-chip resources is difficult to design. In this paper, we propose a method for developing such a controller: evolving a recurrent neural network using the Enforced Subpopulations algorithm. The method is tested in a trace-based simulation that measures dynamic assignation of a pool of level-two cache banks to a set of processing cores. We present results showing that, when the chip is controlled by the neural network, we obtain a 13% performance improvement over static cache partitioning.

1 Introduction

Scalability limitations on the performance growth of conventional superscalar designs–due to growing on-chip wire delays [1]–will make chips with many processors per chip nearly universal. These chip multiprocessors, or CMPs, have already started to emerge: IBM will soon ship their Power4 processor [4], which has two processing cores per die, and Compaq is developing an eight-core chip called Piranha [2].

An open question is how the cache hierarchy and off-chip interconnects will be designed as the number of cores on a chip increases from two to eventually hundreds. It is likely that each core will have a private level-one cache, which will be small and tightly coupled to its processing core. The level-two (L2) caches, however, will consume much of the die, reducing the frequency with which processors must go off the chip for data. These L2 caches will total tens (and eventually hundreds) of megabytes in size, but will be distributed across the chip, and divided into hundreds of physical banks.

In the simplest implementation, each core is assigned its dedicated memory resources (i.e. some number of L2 cache banks) at design time. However, such a static assignment will render performance suboptimal, as different workloads have different memory requirements, as well as resource needs that vary over time. For instance, if job A is only using a small fraction of its L2 cache, and job B is memory bound, then performance would be improved by dynamically assigning some of the cache banks from job A's processor to job B.

If a sufficiently effective (and low-overhead) mechanism can be found to manage these resources adaptively in response to the changing needs of the jobs running on the individual cores, performance could be significantly improved. Resources could be allocated to where they contribute most to maximizing some desired measure of overall chip performance (figure 1). The controller would need to use the available CMP state information to periodically re-assign cache banks to the cores and minimize the number of L2 cache misses. accesses).

Unfortunately, the design of such a controller is challenging for the following reasons:

1. Conventional controllers need to implement an analytic model, which may not accurately represent the true behavior of the system, or it may be too computationally intensive to produce resource management recommendations in time. If inaccurate, it is unclear *a priori* what effect a control action will have on the future behavior of the system.
2. The system has many competing dimensions (power, reliability, application phase, workload mix, memory bandwidth, cache size), posing a dif-

[1] To appear in the *Proceedings of the International Joint Conference on Neural Networks*, 2001.

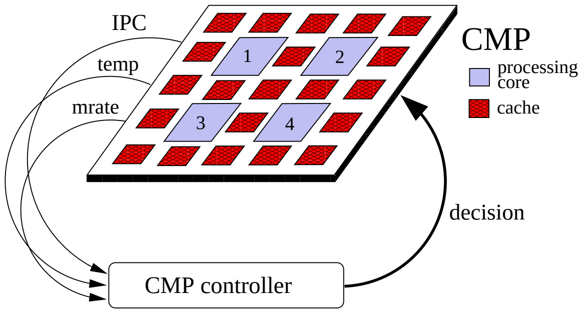

Figure 1: Controlling a Chip Multiprocessor. The controller receives measurements from the chip at regular intervals and outputs a decision to optimize the performance for some desired mode of operation.

ficult challenge to a controller trying to find a locally optimal performance point while satisfying other constraints.

3. The system may be non-Markov, making effective decisions require more information that just the current state of the chip.

In short, the problem exhibits the basic characteristics of difficult reinforcement learning tasks—a sequence of decisions have to be made without prior knowledge of what action should be taken in each state to ensure good long term performance. CMP resource management is thus likely a good candidate for the application of reinforcement learning techniques such as artificial evolution.

In this paper, we propose a method for developing a CMP controller to take a first step toward on-line dynamic CMP resource management (just L2 cache banks), using a neuroevolution algorithm called Enforced Subpopulations (ESP). ESP is used to evolve a neural network that controls the L2 cache size of each individual processing core.

The paper is organized as follows: Section 2 provides general background on Neuroevolution. Section 3 presents the ESP algorithm. Section 4 describes the methodology for evolving the CMP controller and presents experimental results. Finally, section 5 discusses avenues for future work in this area.

2 Neuroevolution

Neuroevolution (NE; see [10] for a comprehensive review of research in evolutionary neural networks) represents a significant departure from conventional reinforcement learning methods based on dynamic programming [9]. Instead of having a single agent learn a *value-function* that indirectly represents its policy, NE searches the space of policies directly using a Genetic Algorithm (GA; [7]). A population of neural networks is encoded into strings of weights called *chromosomes* that represent the *genotype* of each network. Following a process analogous to natural evolution, each genotype is transformed into its neural network *phenotype* and evaluated on a given task to assess its *fitness*. Those networks that receive high fitness are then mated by exchanging genotype substrings to produce new networks. By mating only the most fit individuals, the hope is that the favorable traits of both parents will be transmitted to the offspring resulting in a higher-scoring individual.

GAs can rapidly locate high payoff regions of high-dimensional search spaces, are less susceptible to local minima than single solution methods, and work well on problems where there is little domain knowledge. For these reasons they are well suited for searching the space of neural network parameters. Instead of training a network by performing gradient-descent on an error surface, the GA samples the space of networks and recombines those that perform best on the task in question.

3 Enforced Subpopulations

Enforced Subpopulations (ESP; [5, 6]) is a neuroevolution method that extends the Symbiotic, Adaptive Neuroevolution algorithm (SANE; [8]) to tasks that require memory. ESP and SANE differ from other NE methods in that they evolve partial solutions or *neurons* instead of complete networks, and a subset of these neurons are put together to form a complete network. In contrast to SANE, ESP makes use of explicit subtasks; a separate subpopulation is allocated for each of the u units in the network, and a neuron can only be recombined with members of its own subpopulation (figure 2). This way the neurons in each subpopulation can evolve independently and rapidly specialize into good network sub-functions.

Evolution in ESP proceeds as follows:

1. Initialization. The number of hidden units u in the networks that will be formed is specified and u subpopulations of neuron chromosomes are created. Each chromosome encodes the input, output, and recurrent connection weights of a neuron with a random string of real numbers.

2. Evaluation. A set of u neurons is selected randomly, one from each subpopulation, and combined to form a neural network. The network is submitted to a *trial* in which it is evaluated on the task and awarded a fitness score. The score is added to the *cumulative fitness* of each neuron

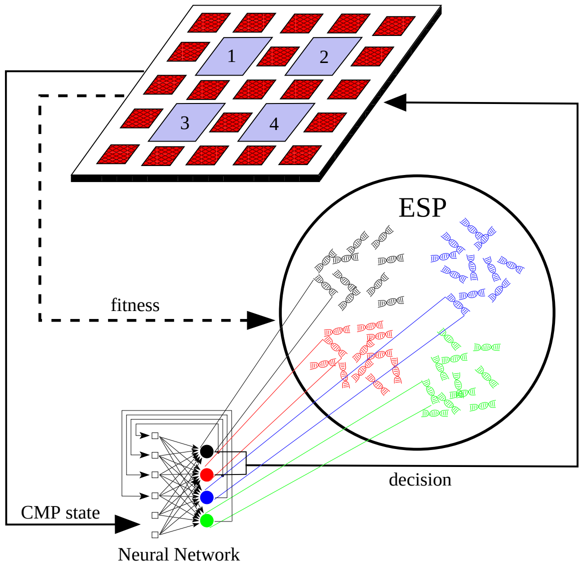

Figure 2: The Enforced Subpopulations (ESP) Method. There is a subpopulation of neurons for each hidden unit position in the recurrent networks being evolved. Networks are formed by randomly selecting one neuron from each subpopulation, and then evaluated by measuring their performance (fitness) in the task. The best neurons are mated within each subpopulation.

that participated in the network. This process is repeated until each neuron has participated in an average of e.g. 10 trials.

3. Recombination. The average fitness of each neuron is calculated by dividing its cumulative fitness by the number of trials in which it participated. Neurons are then ranked by average fitness. Each neuron in the top quartile is recombined with a higher-ranking neuron using 1-point crossover and mutation at low levels to create the offspring to replace the lowest-ranking half of the population.

4. The Evaluation–Recombination cycle is repeated until a network that performs sufficiently well in the task is found.

Evolving networks at the neuron level has proven to be a very efficient method for solving reinforcement learning tasks such as pole-balancing [6], robot arm control, and game playing [8].

ESP extends the neuron-level evolution introduced in SANE to tasks that require memory. Because neurons are segregated into subpopulations they are guaranteed to be combined with one member of each evolving specialization when networks are formed. This reduces the noise in the measurement of neuron fitness and allows recurrent connections between neurons to evolve reliably.

The ability to evolve recurrent networks means that ESP can solve tasks that demand more than a fixed reaction to each sensory input (i.e. reactive behavior). The solutions ESP generates can retain information from previous inputs and use it to make decisions. This property is essential for the CMP control task where the instantaneous state of the chip does not provide enough information about the behavior of jobs (e.g. memory access patterns) to predict future states and make intelligent decisions. By evolving recurrent networks, ESP finds solutions that can identify features in a history of CMP states to determine the underlying, *hidden* state of the CMP that is critical to make the correct decision.

4 Evolving the CMP controller

This section describes the methodology for evolving the CMP controller, and presents the results of our preliminary experiments. For this initial study, we restrict the problem to that of evolving a recurrent neural network to manage the L2 cache resources of a CMP with C processing cores. The network must dynamically set the size of each core's L2 cache so that the total job throughput of the chip is maximized. Because the cores execute tasks that vary widely with respect to working-set size and memory access behavior, the challenge for ESP is to evolve a network then can accommodate the changing needs of the separate tasks so that cache misses are minimized.

The following sections cover the three basics components involved in applying ESP this problem: (1) the choice of input and output representations for the networks, (2) the simulation environment that models the behavior of the CMP, and (3) the evaluation function that measures the relative quality of the interactions between (1) and (2). The results are presented in the final subsection.

4.1 Network Representation

Figure 3 shows the representation used in our experiments. The input layer receives the instructions per cycle (IPC), L1 cache miss rate (L1m), and L2 cache miss rate (L2m) of each core $\{c_i\}_{i=1}^{C}$. These three variables constitute the state of the chip that is observable to the controller, and were chosen intuitively to be the measurements most relevant to L2 cache resizing. Because the networks are recurrent, they also receive the previous hidden layer activation as input, for a total of $3C + u$ input units. The output layer has one unit per core whose activation value is the amount of cache desired for that core. All hidden and output units are sigmoidal.

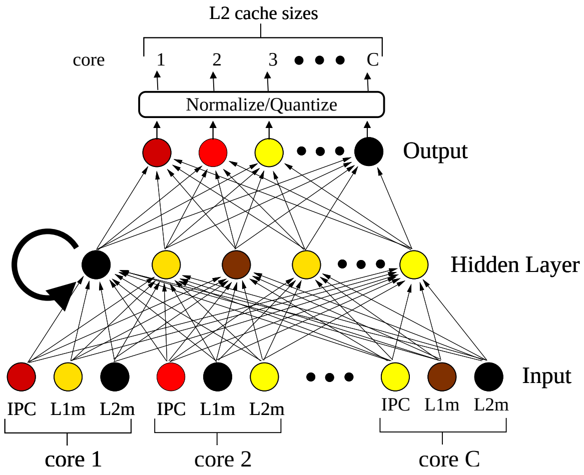

Figure 3: CMP control network. The network has a set of input units for each core, one unit for each of the three performance measurements (IPC,L1m,L2m). There is one output unit per core. The activation of an output unit, indicated by gray-level, corresponds to the amount of L2 cache requested by that core.

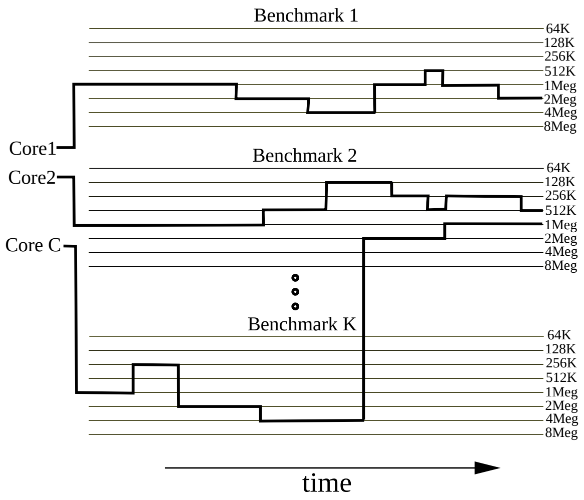

Figure 4: Trace environment. The CMP is approximated by several sets of traces, one for each benchmark. There are as many traces active during a network evaluation as there are cores. There is one trace in each set for each of the cache sizes. When the controller changes a core's cache size the trace environment switches over to the appropriate trace.

To prevent the total amount of cache requested by a network from exceeding the total cache available on the chip ($L2_{total}$), the network output is post-processed by normalizing the activations so that they add up to 1. Because the output units produce continuous values, the normalized activations are then quantized into the number of different cache sizes available (see below).

4.2 Simulation Environment

Controllers were evolved in an approximation to the CMP environment that relies on traces collected from the SimpleScalar processor simulator [3]. A trace is a sequence of measurements of processor variables sampled at fixed intervals, and is a common way to capture various characteristics of processor behavior. A set of traces was generated for each of the following SPEC2000 benchmarks: `art`, `equake`, `gcc`, `gzip`, `parser`, `perlbmk`, `vpr`, using their respective reference working-sets. Each benchmark's trace set consists of one trace for each possible L2 cache size $s \in S = \{64K, 128K, 256K \ldots L2_{total}\}$, for a total of $7 \times |S|$ traces. For convenience, traces are identified by the naming scheme: T<`benchmark name`><`cache size`>. For example, T`gcc256` is the trace for the `gcc` benchmark for a processor with 256K of L2 cache. All traces recorded the IPC, L1m, and L2m of the simulated processor every 10,000 instructions using the DEC Alpha 21264 processor configuration.

The traces provide a substitute for the actual CMP for which a full simulator is not currently available. By combining n traces we can approximate a CMP with n processing cores. Taking the recorded values (IPC,L1m,L2m) from the k-th entry in each of the n traces gives the state that the CMP would be in after $10,000 \times k$ instructions have been executed on each of the cores.

The next section describes how the two components covered so far are combined to evolve the CMP controller.

4.3 Network Evaluation

Networks are evaluated by having them interact with the trace-based environment for some fixed number of control decisions. At the beginning of a network evaluation the environment is initialized by selecting a set of C benchmarks, and allocating an equal amount of L2 cache to each core ($L2_{total}/C$). Once initialized, the network starts controlling the CMP by receiving the state of the chip at time t from the traces corresponding to caches of size $L2_{total}/C$. The network then outputs its cache allocation decision which affects the configuration of the chip from t until the next decision point at time $t+1$, 10,000 instructions later. The next state at $t+1$ then becomes the new input to the network and the cycle is repeated.

In a real CMP, the reassignment of a cache bank from core A to core B would cause the entire caches of A and B to be unavailable for a significant number of cycles while their data is being written back to memory.

In our simulation environment, we ignore this overhead and simply reconfigure the chip by switching to the trace corresponding to the new cache size (figure 4). So, for instance, if the controller decides that core c_1, which is currently executing the `gzip` benchmark with a 256K cache, should have 512K, then the trace for c_1 will switch from $Tgcc256$ to $Tgcc512$, and the controller will receive values from $Tgcc512$ at the next decision point. The new trace is started at the same point as the old one (i.e. the same number of instruction into the computation). When a trace runs out, the environment switches to the trace of a different benchmark at the same cache size (see trajectory of core C in figure 4). The evaluation ends after some predefined number of cycles.

All of the experiments presents here were conducted using the following parameter settings.

Parameter	Value
Environment	
cores (C)	4
caches sizes ($\|S\|$)	7 (64k..4M)
$L2_{total}$	4M
ESP	
number of subpops	10
size of subpops	100
mutation rate	30%

The values for ESP are a compromise between performance (i.e. the quality of the solution) and the CPU time required for the simulation. Larger values produced similar results with a linear increase in CPU time.

With 7 possible cache sizes available to each core and 7 benchmarks, a total of 49 traces were used to implement to environment. Each network was evaluated for 1 billion instructions (i.e. 100,000 decisions The fitness of a network was the average IPC of the chip averaged over the duration of the trial.

Although using this trace-based approach simplifies the CMP environment somewhat, it should provide a good first-approximation with which to evaluate the feasibility of actually applying ESP to a full-scale version of this problem. Furthermore, if the CMP is reconfigured using a much more efficient implementation where the number of cache banks assigned to a core is changed by increasing or decreasing the associativity of its total cache, then our trace-based model more closely approximates the true behavior of the chip.

4.4 Results

Five simulations were run on a 14-processor Sun Ultra Enterprise 5500 for approximately 1000 generations each. At the end of each evolution the fitness of the best network was compared to a baseline performance value

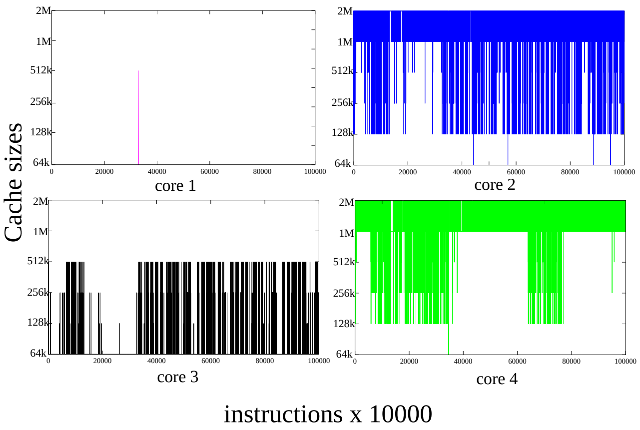

Figure 5: Control Behavior. Each plot shows the cache size of one of the 4 cores over the course of 1 billion instructions.

that is the average IPC of the chip obtained when the total amount of L2 cache on the chip is divided equally among the cores. The networks showed an average improvement of 16% over the baseline.

Although a significant improvement, this measure of performance is inadequate to determine how well the networks are able to allocate cache resources in general. The reason for this is that the networks were evaluated in a single trial and, therefore, only tested in one set of conditions. The danger with this approach is that the networks will specialize to these particular conditions and not capture much of the general behavior. One way to avoid this is to evaluate each network in many different contexts. That is, many trials in which the set of benchmarks run by the CMP are chosen at random. While a possible alternative in this case, time constraints prevented the use these more costly, multiple evaluations.

Instead, to measure how well the networks perform the general task despite their limited exposure to the environment, the best network from each of the 5 simulations was submitted to a generalization test. The test consists of 1000 trials where the network controls the chip for 1 billion instructions under random initial conditions. In each trial, the baseline performance was also measured. Once all the trials were completed, the network performance was compared to the baseline across all trials. The result of this test showed that the networks still retained a 13% average performance advantage over the baseline, and, perhaps more importantly, the networks performed better than the baseline on every trial. These tests show that although the networks had very limited exposure to the task during evolution, they were able to extract general competence to perform well under novel circumstances.

Figure 5 shows the behavior of one of the best networks over the course of 100,000 decisions. It is clear that the different cores that are running different benchmarks are being managed differently. Core c_1 stays almost entirely at $64K$ while the others oscillate rapidly within characteristic ranges. This oscillation is an artifact of not imposing an overhead on cache re-sizing.

5 Discussion and Future Work

The results so far indicate that evolutionary neural networks can potentially provide significant improvement over a simple equipartition of the L2 cache on a chip with 4 cores. It remains to be seen whether it will scale to larger architectures. Further studies will begin by investigating this scalability question, and comparing the approach to other possible, heuristic methods.

The trace-based model currently ignores the following characteristics of the CMP microarchitecture. (1) As mentioned in the previous section, the absence of a reconfiguration penalty allows the controller to "over-manage" the resources with impunity, (2) it treats all cache banks as if they are equidistant from each of the core without accounting for the variability in cache access latencies that exist due the physical layout of the chip—some banks are necessarily further from a core and require more cycles to access, and (3) to reduce the number of traces in the model, the cache sizes available to each core grow in powers of 2 ($2^n \times 64K$), instead of linearly ($n \times 64K$). This limits control to a relatively coarse partitioning of the cache.

A model that incorporates these complexities will force ESP to evolve controllers that are more reserved in their resource management regime, favor cache banks that reside at close proximity to the cores, and are able to control resources at a finer granularity.

Once larger and more accurate simulations have been conducted the viability this approach should be more firmly understood. If further study proves encouraging, the next step will be to start looking a hardware implementation issues for the neural network controller. For the controller to be useful it must produce timely decisions, which means that it will likely need to be implemented hardware. The cost of putting controller on-chip will have to be weighed against the potential performance improvement provided by this additional hardware.

6 Conclusion

The approach to CMP control presented in this paper is a first step toward the complex resource management problem that will be critical as large-scale multiprocessor become widespread. Our initial results are promising, and the approach should be well suited the the very high-dimensional state-space implied by multiprocessors with many dynamically adjustable parameters and modes of operation.

Acknowledgments

This research was supported in part by the National Science Foundation under grant IIS-0083776. Special thanks to Hrishikesh Murukkathampoondi for his assistance in setting up the SimpleScalar parameters, generating the traces, and overall advice.

References

[1] Agarwal, V., Murukkathampoondi, H., Keckler, S., and Burger, D. (2000). Clock rate versus IPC: The end of the road for conventional microarchitectures. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*.

[2] Barroso, L. A., Gharachorloo, K., McNamara, R., Nowatzyk, A., Qadeer, S., Sano, B., Smith, S., Stets, R., and Verghese, B. (2000). Piranha: A scalable architecture based on single-chip multiprocessing. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, 282–293.

[3] Burger, D., and Austin, T. M. (1997). The simplescalar tool set version 2.0. Technical Report Technical Report 1342, Computer Sciences Department, University of Wisconsin.

[4] Diefendorff, K. (1999). Power4 focuses on memory bandwidth. *Microprocessor Report*, 13(13).

[5] Gomez, F., and Miikkulainen, R. (1997). Incremental evolution of complex general behavior. *Adaptive Behavior*, 5:317–342.

[6] Gomez, F., and Miikkulainen, R. (1999). Solving non-Markovian control tasks with neuroevolution. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*. Denver, CO: Morgan Kaufmann.

[7] Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Ann Arbor, MI: University of Michigan Press.

[8] Moriarty, D. E. (1997). *Symbiotic Evolution of Neural Networks in Sequential Decision Tasks*. PhD thesis, Department of Computer Sciences, The University of Texas at Austin. Technical Report UT-AI97-257.

[9] Sutton, R. S., and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press.

[10] Yao, X. (1993). A review of evolutionary artificial neural networks. *International Journal of Intelligent Systems*, 8(4).