Application-Specific Heterogeneous Multiprocessor Synthesis Using Differential-Evolution

Allan Rae, *Student Member, IEEE* Sri Parameswaran, *Member, IEEE*
Department of Computer Science and Electrical Engineering
University of Queensland, St. Lucia, Queensland, Australia.
{rae,sridevan}@csee.uq.edu.au

Abstract

This paper presents an application-specific, heterogeneous multiprocessor synthesis system, named HeMPS, that combines a form of Evolutionary Computation known as Differential Evolution with a scheduling heuristic to search the design space efficiently. We demonstrate the effectiveness of our technique by comparing it to similar existing systems. The proposed strategy is shown to be faster than recent systems on large problems while providing equivalent or improved final solutions.

1. Introduction

There is a growing need for the automated design of application-specific multiprocessor systems. Such systems range from high-performance signal processing systems, robot arm controllers to video-cameras and automobiles. Even modern "uniprocessor" workstations make use of multiple processors to achieve maximum performance. These additional processors are specifically designed to handle I/O intensive tasks such as peripheral interfaces, for example SCSI-controllers and graphics processors. In each case the system is a heterogeneous multiprocessor, hereafter referred to as HeMP, because they are constructed from multiple different processor types. It is therefore important to devise systematic methods of designing such systems.

In this paper we investigate a strategy for the synthesis of application-specific HeMP systems. This strategy, designated HeMP-Synthesis (HeMPS), synthesises a distributed, heterogeneous multiprocessor architecture, allocates subtasks to each of the processors in the system and provides a static task execution schedule. HeMPS does this by combining a problem-specific heuristic with a form of evolutionary computing known as Differential Evolution.

The next section describes the previous research on problems related to the synthesis of multiprocessor systems. Section 3 provides a brief introduction to the operation of differential evolution. Section 4 describes our synthesis strategy in detail. Section 5 describes some examples and experimental results and provides a comparison of our proposal against some existing synthesis systems.

2. Previous Work

The automated synthesis of heterogeneous multiprocessors is a relatively young field that incorporates the scheduling and allocation of tasks with the selection of processors to execute those tasks. Scheduling and allocation of tasks on multiprocessors has been studied extensively with most scheduling schemes assuming a fixed homogeneous multiprocessor as the target system. The scheduling and allocation problem has been shown to be NP-complete [10] so a variety of heuristics have been developed. Many of these heuristics have been compared by researchers such as Khan et al [13] who classified and compared three types of multiprocessor scheduling heuristics: critical path heuristics, list scheduling heuristics and graph decomposition methods. Most scheduling heuristics have failed to consider communication overhead however some notable exceptions include: Mapping Heuristic (MH) [7], ELS and ETF heuristics [12].

Prakash and Parker were pioneers in HeMP synthesis with *Synthesis of Systems* (SOS) [15]. This system utilised a mixed integer linear-programming (MILP) model. While providing optimal solutions this model was impractical for even small numbers of tasks because of the exponential increase in the number of model factors as the number of subtasks increased. A corresponding exponential increase in execution time resulted. Almost all work since then has concentrated on reducing the search time through improved heuristics and randomised search methods such as genetic algorithms.

The SHEMUS system by Dhodhi, Ahmad and Storer [5] introduced genetic algorithms to the heterogeneous multi-

processor synthesis field. Their system is the most similar to this proposal. The SHEMUS system utilises a problem-space genetic algorithm to search for an optimum list-schedule and the numbers of different processors. An initial solution is determined and the population is initialised by mutating this initial solution.

Wolf introduced a multi-step heuristic algorithm [23] for distributed, embedded computer design. The algorithm generates an initial allocation and mapping to meet the specified deadlines (rate constraints) and then undergoes a process of design refinement to reduce system cost. The refinement steps are: minimise processor cost, reallocate processes to minimise interprocessor communication, minimise communications port cost (by choosing appropriate connection technology) and finally minimise the device cost.

Dasgupta and Karri have proposed algorithms for synthesising reliability-optimal HeMP systems [2]. Their algorithms maximise reliability by selecting processors and communication links that have a high reliability and will achieve the desired time constraints rather than utilising multiple redundant components.

Nguyen and Nguyen proposed an application-specific embedded computer synthesis methodology called Architecture Process Flow [14]. Their proposal describes a semi-automatic process for converting the problem specification, in the form of an existing application program, into a final system. Their proposal suggests the use of several expert systems driven by fuzzy logic or neural networks to aid in topology and component selection.

3. Differential Evolution

Evolutionary Strategies (ES) [17, 18, 19] were originally developed in Germany during the 1960s while a similar technique, Evolutionary Programming (EP) [9, 8], was being developed independently in the United States during the same period. These evolutionary techniques have many similarities to Genetic Algorithms (GA) [3, 11], with the major difference being a greater reliance on mutation in the evolutionary techniques. ES, EP and GA all consist of an evolving population of trial solutions. These trial solutions compete against each other and the better solutions are used to generate a new population. Thus over time the population converges on possibly several local minima. If the population is sufficiently large and initially well spread across the problem space it's likely that the global minima will be found.

Differential Evolution (DE) [22, 16] is an evolutionary descendant of ES. Like ES it attempts to optimise the mutation rate for each parameter of the trial vector. Mutation efficiency is maximized when the mutation step size is comparable to the standard deviation of the parameters in the population. The optimum step size for each parameter is different and also varies over time. DE simplifies the process of mutation by using the scaled difference between two randomly-chosen trial solution vectors, $\vec{X}_a$ and $\vec{X}_b$, as the mutation of a third trial solution vector: the parent vector $\vec{X}_c$. That is, $\vec{X}_c' = \vec{X}_c + F(\vec{X}_a - \vec{X}_b)$ where F is a scaling factor. Some variants of DE also make use of multiple, weighted-difference vectors or use the current best vector as one of the terms in the difference. By generating the mutations from within the population itself DE avoids the complex task of tracking and adjusting the mutation step sizes for each parameter while achieving near optimum mutation. DE also makes use of recombination, otherwise known as crossover in GA literature, which involves mixing in some of the parameters from the original vector.

However, recombination as used in DE is somewhat different to the crossover used by standard GAs. Crossover in GAs involves slicing two parent chromosomes at some predetermined point/s and using alternate slices from each parent to form an offspring. In DE however, recombination occurs in one of two ways: exponential or binomial. Exponential recombination starts from a randomly chosen point in the trial vectors and is controlled by comparing a uniformly distributed random number in the range [0,1] with a crossover constant, CR. If the random number is less than CR the parent vector gets its parameter replaced by the corresponding parameter from the mutated vector and this process continues until either all of the vector has been replaced by the mutated vector or the random number is greater than CR. Thus, exponential recombination applies a single slice from the mutated vector. Binomial recombination starts at the beginning of the vector and replaces the current parameter with the corresponding parameter from the mutated vector if a random number is less than CR. This process continues for the length of the vector.

The variations of DE are classified using a simple notation that is best described using an example. The designation: *DE/best/1/exp*, means the best trial solution so far is used as the parent; one vector difference is used for mutation and exponential crossover is used. Thus the last entry is always used to indicate either *exp*onential or *bin*omial crossover. The second last entry indicates how many vector pairs form the vector difference used in the mutation step. The remaining classifier, the parent classifier, is one of *best*, *rand* or *rand-to-best* – where *rand* indicates a randomly chosen parent and *rand-to-best* indicates the parent is used with the best vector as a difference pair although this difference pair is not counted in the second last entry.

If the modified trial vector yields a lower objective function than the original parent vector then it replaces that vector in the new population, otherwise the old vector is retained. This process of selection is much faster and simpler than that used in most other ES, EP or GA algorithms where

selection involves ranking or competitive trials. The population size, N_p, is recommended to be between five and ten times the number of parameters in a vector.

4. Synthesis Strategy

In this section we will formulate the problem, describe our assumptions and provide an outline of the proposed algorithm and the objective function for synthesising an application-specific HeMP system from a given task flow graph and set of design constraints.

4.1. Problem Formulation

The synthesis process takes as input the application specification in the form of a task flow graph and a set of design constraints; it produces a hardware architecture, a static task execution schedule and a mapping of subtasks to processors. The task flow graph consists of n_s subtasks and provides the dependency and communication relations between subtasks. The design constraints include a library of processors of varying cost and capabilities, and a library of communication link types and their costs and capabilities.

For a given application a table of processor execution times is constructed which describes the time required to execute each of the subtasks on a particular processor from the library. Thus the problem can be formulated as: given a task flow graph and a heterogeneous set of processors generate an optimal multiprocessor system by selecting the type of each processor, finding a point-to-point interconnection pattern between the processors and provide a static schedule of subtasks on the processors.

4.2. Assumptions

We have used the same assumptions used by previous research so our results can be directly compared with their work. These assumptions were first defined by Prakash and Parker [15]:

- The system model consists of a distributed-memory system with point-to-point message-passing. The processors available for selection may have varying functionality, cost and performance.
- The execution of a subtask may only start after all the data from its predecessors has arrived.
- A processor can be executing at most one subtask at any given time.
- Subtask execution is non-preemptive. That is, once a subtask begins execution on a given processor it must run to completion uninterrupted.
- While a processor is executing a subtask it is capable of simultaneously transmitting the computed results of a previous subtask to another processor. That is, computation and communication may overlap.
- Interprocessor communication takes place by message-passing over half-duplex point-to-point communications links. The communication cost and transfer sizes are unity.
- The communication cost between two subtasks executed on the same processor is negligible.

One other assumption is that the execution times for each subtask on each processor have been determined *a priori*.

4.3. Objective Function

There are n processors in the system. Each processor p_i has an associated cost C_{p_i} and if a communication link exists between processors p_i and p_j it has a cost $C_{L(i,j)}$. Thus the total implementation cost, C_t, is shown in equation 1. The total execution time is represented by T_t.

$$C_t = \sum_{i=1}^{n} C_{p_i} + \sum_{i=1}^{n} \sum_{j=1}^{n} C_{L(i,j)} \tag{1}$$

A target value, V_t, can be set along with the choice of whether that target value is a cost or performance figure. HeMPS then attempts to optimise the system to match this target value. For example, if V_t represents a performance target then the effective objective functions are shown in equation 2 below. This represents searching for the lowest cost system with the given target performance of V_t. Alternatively, it is also possible to find the highest performance system for a given target cost.

$$T_t \rightarrow V_t \ , \ C_t \rightarrow 0 \tag{2}$$

4.4. Algorithm Outline

HeMPS uses a problem-space DE (PSDE) which is similar to the problem-space GAs (PSGAs) that have been developed by other researchers [4, 5, 6]. Each trial solution vector represents the number of each of the processor types to use in this solution and a priority list used by a scheduling heuristic. Figure 1 shows an overview of the DE search algorithm.

The wide range of the random numbers used to initialise the population gives initial solutions that have more resources than required but the best chance of finding the optimum performance solution. Any excess processors are removed after the scheduling heuristic has determined how many processors it actually used from those available.

The search may finish early if the population has converged or if there has been no improvement upon the best after some specified number of generations. The scheduling heuristic is shown in Figure 2 and will be described below.

Get the population size (N_p), number of generations (N_g), target (V_t) and cost/performance choice.
Initialise the population X with uniform random values in the range $[0, n_s/2]$ where n_s = number of subtasks.
Determine X member quality with scheduling heuristic.
Record the best solution.
repeat
  foreach parent $\vec{X}_c$ in X
    Randomly select a difference pair $\vec{X}_a$ and $\vec{X}_b$
    Calculate $\vec{X}_c^{'} = \vec{X}_c + F(\vec{X}_a - \vec{X}_b)$
    Calculate $\vec{X}_c^{''} = \mathrm{Crossover}(CR\,,\,\vec{X}_c\,,\,\vec{X}_c^{'})$
    Determine the quality of $\vec{X}_c^{''}$
    if $\vec{X}_c^{''}$ is better than $\vec{X}_c$,
      then $\vec{X}_c^{''}$ goes in new population $X^{'}$
      else $\vec{X}_c$ goes in new population $X^{'}$
  endfor
  New population $X^{'}$ replaces X
  Update the best solution.
until population converged or target reached

Figure 1. Differential Evolution algorithm

Build a priority list from the trial solution
Initialise ready_list
current_time = 1
while task not completed **do**
  Update ready list
  for each subtask t_i on the ready_list (chosen in priority order) **do**
    for each processor P_j which is available
      and on which the subtask t_i can be run **do**
      Determine the execution time of t_i on P_j
    endfor
    Schedule t_i on the fastest available processor P_{fa}
    Allocate communications links
    Remove t_i from the ready_list
  endfor
  Increment current_time
endwhile
T_t = current_time
C_t = Total component and communication link costs
return (T_t, C_t)

Figure 2. Multiprocessor Scheduling and Allocation Heuristic

The mutation, crossover and selection stages have been described previously.

The scheduling heuristic used is a simplified list scheduling algorithm. The scheduler takes as input a task flowgraph, a priority-list of subtasks, and the number of processors of each available type. Its output is the total execution time to complete all the subtasks, total implementation cost and the number of each type of processor used.

The priority list is taken from the trial vector and is used to control the scheduling order. A list of subtasks that are ready to run is also kept. Thus the subtask in the ready list with the highest priority is scheduled first. A restriction introduced by heterogeneous processing is that some processors may not be able to execute certain subtasks. As such, it may be impossible to schedule a given priority list if a subtask has a higher priority than another subtask it depends upon. It is therefore necessary to check that a proposed schedule is achievable and if it is not achievable a simple repair procedure is used that swaps the first blocked subtask with the first subtask that it depends upon that appears after it in the list. The repair process iterates until the priority list specifies an achievable schedule.

If multiple free processors exist then we select the processor that will execute the subtask in the minimum time. By allocating the subtask to the fastest available processor the total time for the task is kept to a minimum for this priority list. If there are multiple "fastest" processors then select the processor with the minimum communication cost. If there are still multiple processors to choose from then select the processor with the lowest utilisation.

4.5. Parameter Tuning

There are only three parameters that control differential evolution. These parameters being the population size (N_p), crossover constant (CR) and the difference multiplier (F). The only other variable is the actual choice of DE variant. Several tests have been conducted to determine which of 20 different variants provide the most consistent results. The best performing DE variants are those that make use of the current best trial vector either as a parent or as part of a difference vector. That is, *best* and *rand-to-best* respectively.

Using the problem-specific heuristic allows a reduction in population size compared to more general ES, EP and GA searches. The originators of DE [22] recommend a population size of five to ten times the number of parameters in a vector. In our experiments we tested population sizes ranging from one to 10 times the vector size. A population size from two to five times the vector size provides a near-optimal or optimal result in 10-30 generations. Larger population sizes also found satisfactory solutions however they also require longer execution times due to the increased number of evaluations per generation. The worst case number of generations was 50-60 generations.

The crossover constant, CR, was varied from zero to 1.0 in steps of 0.1 and the difference multiplier, F, was varied from 0.2 to 1.2 in steps of 0.1. For small problem sizes almost every combination of F and CR tested found the optimal solution. However, for larger problems a high F generally required a low CR and vice versa. An example of a consistently performing DE variant and parameter settings is: DE/best/2/bin with $F = 0.2$ and $CR = 0.8$ and a

population size of four times the size of a trial vector.

5. Results

We have implemented HeMPS in approximately 1600 lines of C code incorporating 20 variants of DE which have been used for testing. The examples we used for testing our algorithm are from related HeMP synthesis research. The examples include Prakash and Parker's examples 1 and 2 (*pp1* and *pp2*) [15] and the *robot*-arm manipulator example of Dhodhi, Ahmad and Storer [5]. The remaining examples are the *juice* [21], *cfuge* [1] and *dye* [20] examples used by Wolf. The component cost, execution times and technology parameters used are those estimated or measured by the respective authors. The times for HeMPS and SHEMUS were recorded on Sun SPARCstation-10 workstations. The times for the proposal by Wolf were recorded on SGI Indigo workstations. The runtimes for SOS were recorded on a Solbourne Series 5e/900. The performance of a synthesised system is represented by the time period of the example task. Thus higher performance corresponds to a lower period. In all cases the target used for HeMPS is the time period. The resulting search is for a cost-optimal implementation with the target performance. Test results are shown in Table 1.

The results for the two examples by Prakash and Parker correspond to their point-to-point interconnection experiments. Example *pp1* has input and output from a subtask during the subtasks execution. HeMPS restricts such communication to the beginning or completion of a subtask respectively. It is for this reason that HeMPS is not able to match the results of the MILP method. In a simple experiment, the four-subtask problem was rewritten so that no communication to or from a subtask takes place during the execution of that subtask. This transformation involved breaking each subtask into four smaller subtasks. HeMPS was then able to synthesise the optimal period 3 and period 2.5 systems found by Prakash and Parker.

For example *pp2* all systems found the set of optimal solutions, for each of the periods their results were published for, with one exception by Wolf's algorithm. The relative increase in CPU times from *pp1* to *pp2* demonstrates the efficiency of the DE search method used in HeMPS. This efficiency is due to the rapid convergence offered by this method.

A comparison of HeMPS and Wolf's multi-step algorithm is also provided in Table 1. The systems synthesised by HeMPS are identical to those synthesised by Wolf's algorithm with the exception of the *juice* example. For the *juice* example, HeMPS finds a single-processor solution while Wolf's algorithm synthesises a two-processor system. The heuristic used by HeMPS will not allocate a subtask to a new processor if the resulting communications overhead results in an execution time longer than that which would occur if the task was run on the original processor. That is, HeMPS refuses to synthesise an expensive, slow system when there exists a cheaper, faster solution. The data for the *juice* example is such that there is no need to use the second processor to meet the deadline – in fact, the system generated by HeMPS will complete at a rate of 2.15 milliseconds – the second processor and serial communications effectively extend the execution time by increasing the communication time. A better implementation, implied by the HeMPS result, would be to use another subtask to pad out the time or switch to an interrupt-driven model.

The *robot* entry in Table 1 compares HeMPS with the SHEMUS system by Dhodhi et al [5]. This example has 25-subtasks and would be impractical to test with the MILP modelling used by Prakash and Parker. Two tests were conducted with HeMPS. The first test was to find the lowest-cost, performance-optimal implementation and the second test was to find the lowest-cost implementation with the same period as that found by SHEMUS. HeMPS succeeded in finding the global optimum in the first test and achieved a considerable improvement in implementation cost for the second test. Unfortunately, the authors have been unable to determine the target used by SHEMUS however we have assumed that maximum performance was the target. Alternatively if the target was the best balance of performance and cost HeMPS is still superior.

6. Conclusions

In this paper, we have presented a strategy for the synthesis of application-specific heterogeneous multiprocessors that use a point-to-point interconnection network. This strategy, HeMPS, combines a fast, problem-specific heuristic with a form of evolutionary computation known as differential evolution. This combination enables the HeMPS system to rapidly and efficiently search the design-space for an optimal or near-optimal solution.

Furthermore, HeMPS has been compared to similar systems and found to provide equal or better results for the examples described. The results of the comparison show that for small numbers of subtasks the execution time of HeMPS is on par with other recent systems. When the number of subtasks is increased the execution time of HeMPS remains low.

The use of heuristic or randomised search methods by a number of authors has enabled considerable reductions in the search and synthesis time. There remains a considerable amount of work before such systems reach the ultimate goal of complete system specification integrating component selection and board layout. However, we believe the HeMPS methodology is an important step towards that goal.

example	#subtasks	period	Implementation Cost				CPU Time (sec)			
			HcMPS	Wolf	SHEMUS	P&P	HcMPS	Wolf	SHEMUS	P&P
pp1	4	2.5	-	14	-	14	-	0.05	-	11
		3	14	14	-	13	0.09	0.05	-	24
		4	7	7	-	7	0.09	0.05	-	28
		7	5	5	-	5	0.09	0.05	-	37
pp2	9	5	15	15	15	15	0.24	0.7	1.3	3732
		6	12	12	-	12	0.16	1.1	-	26710
		7	8	8	-	8	0.16	1.6	-	32320
		8	7	8	7	7	0.18	1.0	1.1	4511
		15	5	5	-	5	0.12	1.1	-	385012
cfuge	3	0.1	17	17	-	-	0.08	0.1	-	-
juice	4	0.1	27	41	-	-	0.08	0.1	-	-
dye	15	0.1	59	59	-	-	0.83	7.2	-	-
robot	25	20	14	-	-	-	1.55	-	-	-
		23	9	-	17	-	1.55	-	7.3	-

Table 1. Comparison of results for examples from Prakash and Parker, Wolf and SHEMUS

Acknowledgment—The authors would like to thank Wayne Wolf for providing the data for the *cfuge*, *dye* and *juice* examples.

References

[1] J. P. Calvez. *Embedded Real-Time Systems: A Specification and Design Methodology*. Wiley, New York, NY, 1993.

[2] A. Dasgupta and R. Karri. Optimal Algorithms for Synthesis of Reliable Application-Specific Heterogeneous Multiprocessors. *IEEE Trans. on Reliability*, 44(4):603–613, December 1995.

[3] L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.

[4] K. Dhodhi. *Data path synthesis using concurrent scheduling and allocation based on problem-space genetic algorithms.* PhD thesis, Dept. of Electrical Engineering and Computer Science, Lehigh University, Bethlehem, PA, August 1992.

[5] M. K. Dhodhi, I. Ahmad, and R. Storer. SHEMUS: synthesis of heterogeneous multiprocessor systems. *Microprocessors and Microsystems*, 19(6):311–319, August 1995.

[6] M. K. Dhodhi, F. H. Hielscher, R. H. Storer, and J. Bhasker. Datapath synthesis using a problem-space genetic algorithm. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 14(8):934–944, August 1995.

[7] H. El-Rewini and T. G. Lewis. Scheduling parallel program tasks onto arbitrary target machines. *Journal of Parallel and Distributed Computing*, 9(2):138–153, 1990.

[8] D. B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, Piscataway, NJ, 1995.

[9] L. J. Fogel. Autonomous Automata. *Industrial Research*, 4:14–19, 1962.

[10] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP Completeness*. W. H. Freeman, San Francisco, CA, 1979.

[11] D. E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley Pub. Co., Reading, MA, 1989.

[12] J. J. Hwang, Y. C. Chow, F. D. Anger, and C. Y. Lee. Scheduling precedence graphs in systems with interprocessor communication times. *SIAM Journal of Computing*, 18(2):244–257, April 1989.

[13] A. A. Khan, C. L. McCreary, and M. S. Jones. A Comparison of Multiprocessor Scheduling Heuristics. Technical Report CSE94-02, Dept. of Comp. Sci. and Eng., Auburn University, Auburn, AL, 1994.

[14] A. Nguyen and C. Q. Nguyen. Using Fuzzy Logic to Derive an Optimum Embedded Application Specific Architecture. In *WESCON 95*, pages 618 –625, 1995.

[15] S. Prakash and A. C. Parker. SOS: Synthesis of Application-Specific Heterogeneous Multiprocessor Systems. *Journal of Parallel and Distributed Computing*, 16:338–351, 1992.

[16] K. Price and R. Storn. Differential Evolution. *Dr. Dobb's Journal*, pages 18–24, April 1997.

[17] I. Rechenberg. Cybernetic Solution Path of an Experimental Problem. Lib. Trans. 1122, Royal Aircraft Establishment, Farnborough, 1965.

[18] H. P. Schwefel. Kybernetische Evolution als Stratgie der Experimentellen Forschung in der Stromungstechnk. Diploma Thesis, Tech. Univ. of Berlin, 1965.

[19] H. P. Schwefel. *Numerical Optimization of Computer Models*. John Wiley, Chichester, U.K., 1981.

[20] B. Selic, G. Gullekson, and P. T. Ward. *Real-Time Object-Oriented Modeling*. Wiley, New York, NY, 1994.

[21] S. Shlaer and S. J. Mellor. *Object Lifecycles: Modeling the World in States*. Yourdon, New York, NY, 1992.

[22] R. Storn and K. Price. Differential Evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, International Computer Science Institute, Berkeley, CA, March 1995.

[23] W. H. Wolf. An Architectural Co-Synthesis Algorithm for Distributed, Embedded Computing Systems. *IEEE Trans. VLSI*, 5(2):218–229, June 1997.