RAPID DESIGN SPACE VISUALISATION THROUGH HARDWARE/SOFTWARE PARTITIONING

Simon A. Spacey, Wayne Luk, Paul H.J. Kelly and Daniel Kuhn

Department of Computing
Imperial College
London, UK

ABSTRACT

This paper introduces the 3SP Design Space Exploration System. 3SP automatically quantifies acceleration opportunities for programs across a wide range of heterogeneous architectures to allow designers to identify promising implementation platforms before investing in a particular hardware/software codesign. 3SP uses a novel program execution model to integrate comprehensive hardware characteristics including clock speed, number of execution units, issue rates, bandwidths and latencies with software program execution, parallelism, control and data flow measurements to estimate codesign performance for evaluating opportunities for hardware acceleration.

1. INTRODUCTION

The 3S Partitioner (3SP) uses 3S [1] program characterisation measurements to automatically partition software for execution on a range of heterogeneous computational architectures to allow rapid design space visualisation and opportunity exploration before committing to a particular implementation platform. 3SP partitions software at the binary level and can be used to generate design curves for any program written in any compilable language and produces a list of code assignments to assist the designer in their initial hardware/software partitioning decisions.

The main contribution delivered by 3SP is a novel high-quality heuristic to quantify partition opportunities for a wide range of architectures. Unlike previous work, the 3SP heuristic is generally applicable and can be seamlessly applied to architectures with superscalar out-of-order CISC processors and architectures with tightly and loosely coupled reconfigurable components.

This paper begins with a brief overview of related work in section 2. In section 3 the 3SP methodology is disclosed and in section 4 results are presented that demonstrate the use of the 3SP system to identify acceleration opportunities for several benchmarks for a range of potential heterogeneous platforms. The paper continues with a discussion of future work and conclusions in sections 5 and 6.

2. RELATED WORK

Table 1 compares the features of the architecture neutral 3SP timing estimation heuristic against the heuristics of previous automatic hardware/software partitioning research.

Characteristic	[2]	[3]	[4]	[5]	3SP
block size	✓	✓	✓	✓	✓
block iterations	✓	✓	✓	✓	✓
data flow	✗	✗	✓	✓	✓
parallel execution slots	✗	✗	✗	✓	✓
communication bandwidth	✗	✗	✗	✓	✓
communication latency	✗	✗	✗	✗	✓
control flow	✗	✗	✗	✗	✓
execution cycle measurements	✗	✗	✗	✗	✓

Table 1. Software characterisation metrics used in previous heterogeneous partitioning heuristics.

In Stitt *et al.* [2], RISC binaries are decompiled and partitioned using loop iteration count profiling. By analysing program binaries, the Stitt method has the advantage of being applicable to any compilable language, however the use of only block size and iteration counts in the partitioning heuristic means the method has limited applicability and is most appropriate for simple single cycle ALUs and tightly coupled architectures.

In Lysecky *et al.* [3], single loop kernels from RISC binaries are partitioned to reconfigurable logic using hardware loop profiling, on-chip CDFG analysis and warp processor technology mapping. The Lysecky approach has the benefit of providing low overhead profiling results through dedicated hardware however, like Stitt [2], the approach uses loop iterations rather than actual timing measurements to identify partition targets which limits the applicability of the approach for architectures with variable-cycle superscalar CPUs.

In Stitt *et al.* [4], RISC binaries are decompiled and partitioned with a greedy algorithm using execution loop profiling and statically determined advanced CDFG information. The Stitt approach has the benefit of reducing data flow costs through shared data analysis but is focused on tightly

coupled architectures and does not take into account architecture communication latencies and bandwidths which are required to partition for distributed hardware components.

In Atasu *et al.* [5], source code is partitioned using a knapsack algorithm based on execution time estimation and data flow requirements. Using source code has the disadvantage of potentially excluding commercial applications where source code is not available, however by including bandwidth information, the Atasu approach has the advantage of being applicable to distributed architectures with significant communication bandwidth constraints.

Other relevant research includes automatic software partitioning [6–14], manual partitioning utilities [15, 16] and program analysis systems [1, 17–19].

3. METHODOLOGY

3SP uses a unique combination of hardware and software characteristics to quantify the acceleration potential of a program for a range of architectures. The information 3SP uses in its heuristic is summarised in table 2 below.

Hardware Characteristic		Software Characteristic	
τ_l	cycle time	execution time	μ_{pr}
ω_l	parallel execution units	parallel execution slots	ϕ_{pl}
ϵ_l	execution efficiency	program code unit iterations	ι_p
λ_{lm}	bus latency	control flows	χ_{pq}
β_{lm}	bus bandwidth	data flows	η_{pq}
Z_l	hardware size capacity	size of code at each location	z_{pl}

Table 2. Hardware and software characteristics used by the 3SP execution model.

3SP obtains the hardware information it requires from a user initialised base configuration file which 3SP automatically sweeps over a range of values while calculating the acceleration opportunities for a design-space. The current 3SP implementation operates at the program basic block level using software characterisation information obtained from 3S tools [1]; however the 3SP approach can be used to partition at any level of program granularity including the functional level provided the above software characteristic information is available. The 3S tools currently used are: 3S hotspot for CPU block-level execution timing, iteration counts and size measurements, 3S parallelism for block level parallelism information, 3S callgrind for control flow information and 3S data_flow for inter-block data flow information.

3SP unifies the hardware and software characteristic information into a single execution time estimate for a set of potential assignments of code sections p to locations l in an architecture. The heuristic is then used to select the best assignment of a program's code sections to particular architectures using the partitioning algorithm described later in this section.

The general 3SP timing estimate heuristic is intuitive and represents the sum of execution times μ_{pl} plus the sum of all communication times c_{pqlm} for a program's code sections assigned to an architecture's locations:

$$t = \sum_{p} \mu_{pl} + \sum_{pq} c_{pqlm} \tag{1}$$

where p, q are code section indices and l, m are location indices and the execution μ_{pl} and communication c_{pqlm} times are defined by equations (2) and (3) below with reference to the architecture characteristics of table 2.

$$\mu_{pl} = \frac{\iota_p \phi_{pl} \tau_l}{\epsilon_l} \tag{2}$$

$$c_{pqlm} = \chi_{pq} \lambda_{lm} + \frac{\eta_{pq}}{\beta_{lm}} \tag{3}$$

The ϵ_l parameter of equation (2) is a hardware implementation efficiency factor that can be considered the maximum sustainable issue rate per execution unit at hardware location l and could be quoted at the program code section level of granularity if required. Equation (2) can be used as a substitute for the hotspot CPU cycle time measurements μ_{pr} if the reference component r where the 3S measurements are made is not part of the heterogeneous architecture being modelled.

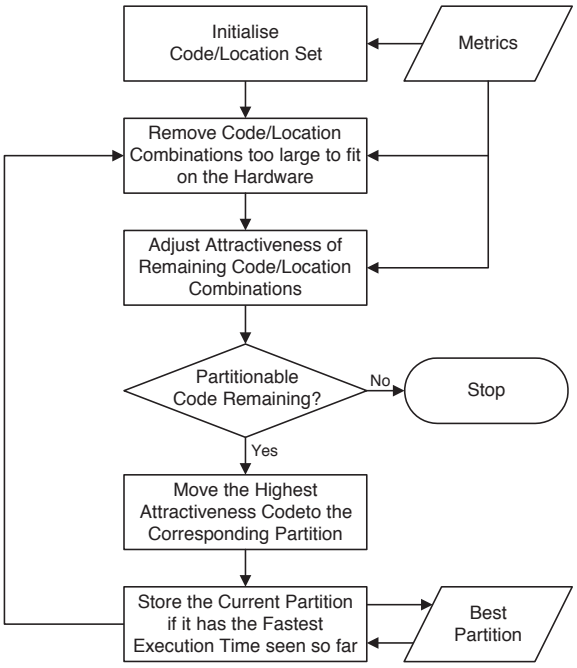

Fig. 1. The Attractiveness Partitioning Algorithm (APA).

3SP uses the Attractiveness Partitioning Algorithm (APA) depicted in figure 1 to rapidly generate high-quality partitioning solutions that minimise the 3SP execution time estimates for code assignments to hardware. The APA algo-

rithm is a heuristic regret minimisation algorithm that partitions considerably faster than a theoretically optimal approach while retainig partition quality with high average partition speedups as discussed in section 5.

The APA algorithm begins with all code sections assigned to an initial reference partition, calculates the 3SP times for each code section if moved to each of the alternate hardware locations in isolation, generates the attractiveness metrics and moves the reference code section with the highest attractiveness (corresponding to the highest potential regret) to it's optimal hardware partition. APA then continues varying the 3SP times to account for the modified cross partition communication costs, recalculating the attractiveness measures and moving the most attractive reference code section to its optimal hardware location until no reference nodes that can fit on an alternate hardware location exist. At each iteration APA keeps a record of the 3SP execution time estimate for the current partition assignments and APA returns the partition assignments with the minimum 3SP execution time observed on algorithm completion.

The attractiveness measure used by APA is the maximum potential single step 3SP speed-up opportunity lost if the program code section p currently located at r is not moved to location l divided by the hardware space requirement z_{pl} of block p on l defined as:

$$\alpha_{pl \neq r} = \frac{min(\{t_{px} \mid x \in locations \backslash \{l\}\}) - t_{pl}}{z_{pl}} \quad (4)$$

for a two component architecture equation (4) simplifies to:

$$\alpha_{pl \neq r} = \frac{t_{pr} - t_{pl}}{z_{pl}} \quad (5)$$

where t_{pr} is the 3SP execution time estimate for the partition with p assigned to an initial reference partition r and t_{pl} the 3SP execution time estimate for the partition with p assigned to the alternate hardware location l. The simplified two component $\alpha_{pl \neq r}$ attractiveness ratio is reminiscent of the regret minimisation Sharpe ratio used in financial portfolio optimisation where a profit difference is divided by a risk [22].

4. RESULTS

This section provides 3SP acceleration results for the synchronous two component heterogeneous architecture presented in figure 2. The design space considered ranges from an SoC architecture tightly coupled through a 32-bit bus operating at the CPU/coprocessor speed, to a loosely coupled architecture with the CPU and reconfigurable coprocessor communicating through a 24x HyperTransport 3.0 bus with the short packet store and forward (S&F) latency characteristics [20], along with a variety of hardware sizes and implementation efficiencies.

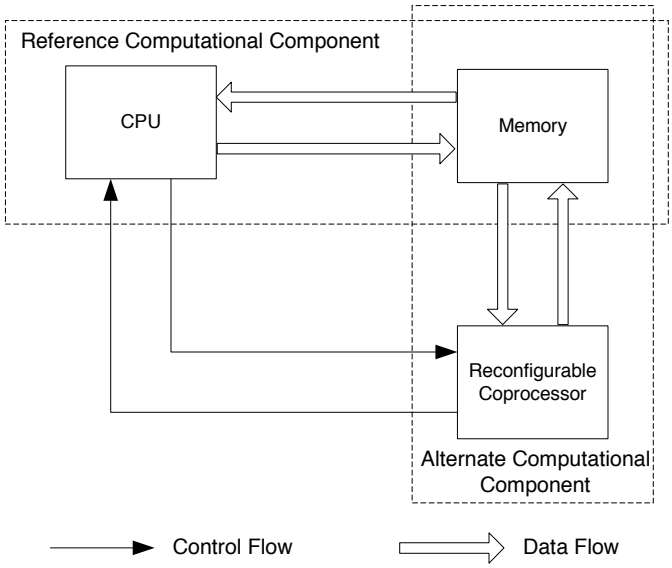

Fig. 2. The heterogeneous computational architecture explored in the results section of this paper.

Results are provided for six MiBench 1.0 [21] benchmarks with actual 3S software execution measurements made on a reference Intel Pentium 4 x86 machine for the MiBench large data sets. The benchmarks are selected to cover the six MiBench categories: crc32 from the Telecommunications class, jpeg (compression) from the Consumer Devices class, stringsearch from Office Automation, sha from Security, susan (smoothing) from Automotive and Industrial Control and dijkstra from the Network class. The smallest benchmark is crc32 with 22 active basic blocks and the largest is jpeg with 1792 active basic blocks. Unless otherwise stated, all results assume a conservative maximum hardware partition size of 256 x86 integer instructions.

4.1. High-Level Acceleration Opportunities

Figures 3 and 4 compare the 3SP program acceleration results for the six MiBench benchmark binaries compiled with three different `gcc` optimisation levels. Figure 3 provides results for a tightly coupled architecture and figure 4 for a loosely coupled architecture.

The figures show that 3SP identifies accelerating partitions with up to 33 times speed-up potential for the benchmarks considered. The impact of the different compiler options are clearly visible with `-O1` having a lower acceleration potential than either `-O2` or `-O3` for some benchmarks supporting the results of [6]. A designer can use these reports to obtain a high level opportunity assessment for a program before deciding whether the acceleration potential justifies further analysis and, if so, which binary and architecture promises to deliver the best return on investment.

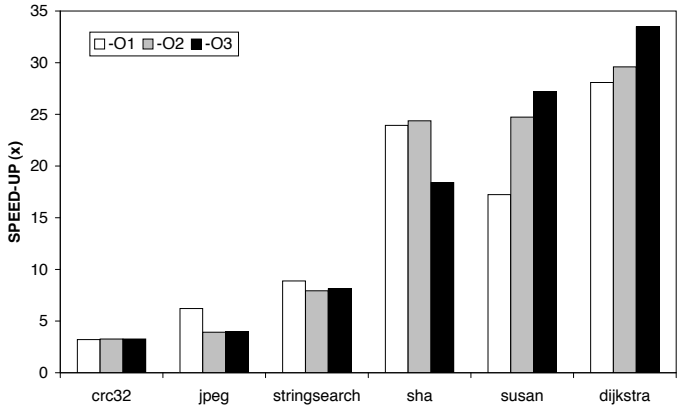

Fig. 3. 3SP program accelerations for MiBench benchmarks compiled with different `gcc` compiler optimisation levels on a 32-bit bus SoC architecture operating at 345MHz.

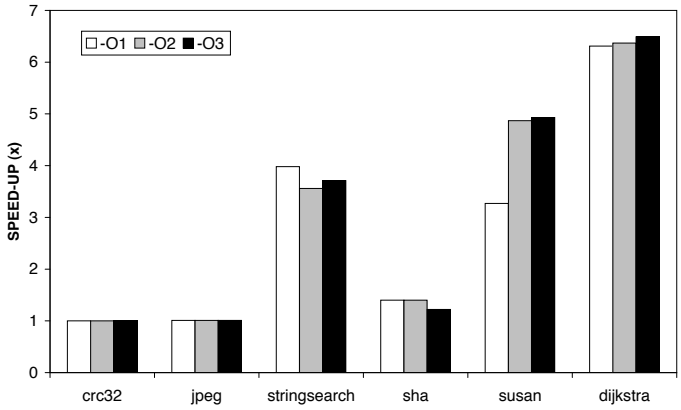

Fig. 4. 3SP program accelerations for MiBench benchmarks compiled with different `gcc` compiler optimisation levels on a loosely coupled distributed architecture with a 2GHz CPU and a coprocessor operating at 345MHz.

4.2. Design Space Exploration

After obtaining a high-level acceleration opportunity report, a designer can use 3SP to delve in detail into the design space and explore the impact of individual or combinations of architecture parameters on program acceleration.

Figures 5, 6 and 7 show the effect of partition size, bandwidth and latency on partition performance for the dijkstra, susan and sha MiBench benchmarks compiled with `-O3` using the tightly coupled architecture report of figure 3 as a starting point. The bandwidth and latency figures cover the full range of architecture characteristics from single chips to distributed architectures, demonstrating the general applicability of the 3SP execution time model.

From figure 5 a designer could conclude that hardware sizes greater than 64 instruction equivalents will not bring significant benefit when accelerating dijkstra in the size range considered. From figure 6 a designer can see that sha is bandwidth sensitive, and from figure 7 the step transitions in 3SP acceleration potential could justify a designer expending extra effort on relative heterogeneous component placement and buffering when attempting to accelerate either the sha or susan benchmarks using 3SP based partitions.

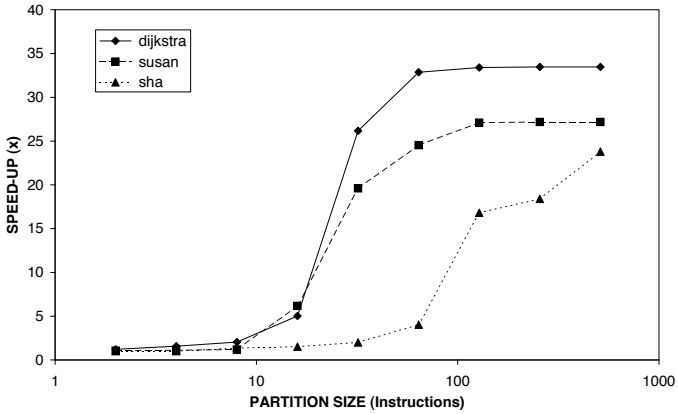

Fig. 5. 3SP program accelerations for MiBench benchmarks compiled with `gcc -O3` on a 345MHz 32-bit bus SoC architecture with hardware of different maximum sizes.

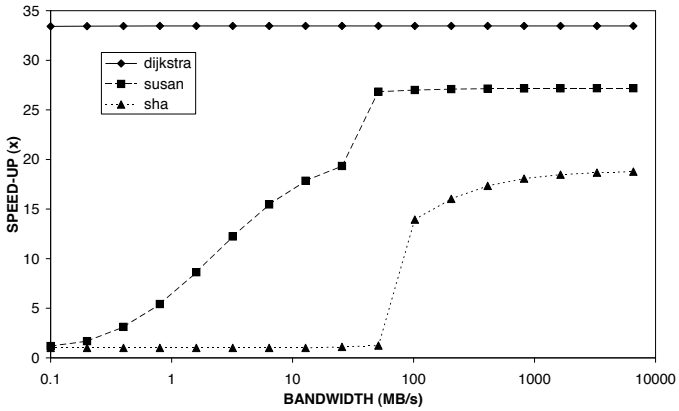

Fig. 6. 3SP program accelerations for MiBench benchmarks compiled with `gcc -O3` on a 345MHz SoC architecture with different CPU to coprocessor bandwidths.

Figures 8 and 9 show the coprocessor clock speed and implementation efficiencies that must be achieved to deliver a required acceleration for the MiBench 1.0 dijkstra benchmark compiled with `gcc -O3` and partitioned using 3SP for a known CPU speed. The curves allow a designer to assess the implementation characteristics required to deliver a desired program acceleration, and can form a basis for hardware cost/benefit analysis.

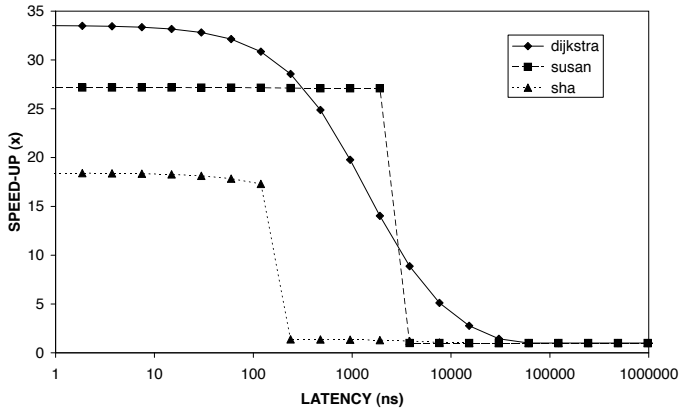

Fig. 7. 3SP program accelerations for MiBench benchmarks compiled with `gcc -O3` on a 345MHz 32-bit bus SoC architecture with different CPU to coprocessor latencies.

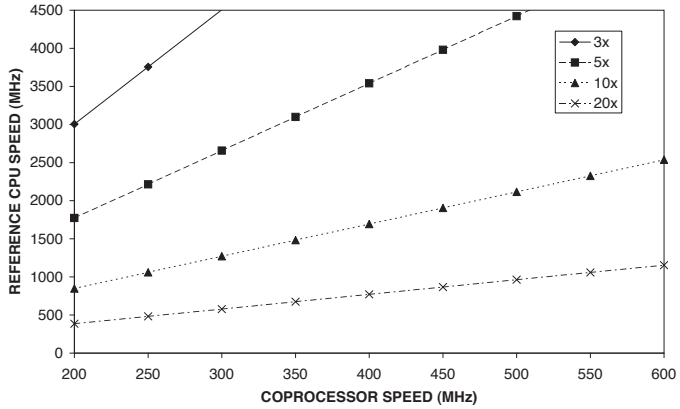

Fig. 8. Maximum CPU speed that can be accelerated to achieve a required 3SP partition speed-up for the dijkstra benchmark compiled with `gcc -O3` using a given coprocessor assuming a 2.89ns latency and 1.28GB/s bandwidth bus.

5. EVALUATION AND FUTURE WORK

The 3SP methodology can be applied to any program partitioning granularity and a range of architecture configurations and components including superscalar CPUs. However, 3SP is currently only implemented for block based partitioning of x86 integer programs because of limitations in the current 3S [1] release (version 2.8) used to gather software characterisation information.

The 3SP hardware/software partitions are generated quickly to allow rapid design space visualisation using the APA heuristic algorithm and are not theoretically optimal for the problem of partitioning with communication costs which is NP-hard [24]. For the design spaces presented in this paper, an unoptimised APA script implementation is up to 16.7 times faster than the optimal CPLEX solver and produces

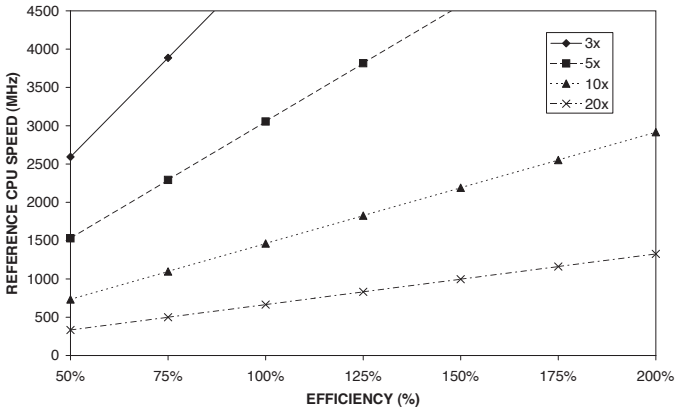

Fig. 9. Maximum CPU speed that can be accelerated to achieve a required 3SP partition speed-up for the dijkstra benchmark compiled with `gcc -O3` using a given hardware implementation efficiency assuming a 345MHz coprocessor with a 2.89ns call latency and 1.28GB/s bandwidth data bus.

partitions with speedups less than 15% below the optimal values on average. The speed of APA can be improved to 31.9 times that of CPELX at an additional 15% loss in optimality by stopping APA on the first negative $\alpha_{pl \neq r}$ value.

Apart from the sub-optimality of the APA partitioning algorithm, the performance opportunities and transition points estimated by 3SP could differ from a physical implementation because of:

1. simplifications in the 3SP execution time model,
2. inaccuracies in the 3S measurements,
3. differences between the actual physical implementation and the 3SP model.

However despite these issues, the 3SP design space curves can still be of use to designers for quickly identifying trends and the sensitivity of programs to architectural parameters.

Future work may address the above issues through more complete simulation (for example using the 3S cache_flow tool to account for cache interactions on code section migration), the use of special hardware for actual timing measurements [3] and the provision of alternative partitioning methods and levels of granularity. Additionally the 3SP model could be extended with program code dependant technology mappings and efficiency factors to allow detailed space estimation through calibration [13] and program code section dependent data issue rates [15].

3SP could be further enhanced to implement the partitions it identifies on real hardware. To do this, the 3SP partitions could be readily re-compiled for hardware using existing source code compilers [15, 16] with a pull-based memory architecture [3, 4] and standard execution synchronisation techniques [8, 9].

6. CONCLUSION

This paper presents an overview of the 3SP Software Partitioning System. 3SP partitions software for execution on a heterogeneous architecture and allows designers to explore the hardware/software codesign before committing to a particular hardware platform. 3SP uses a novel heuristic that combines actual software size, run-time, parallelism, control and data flow measurements with hardware characteristics to allow seamless design space exploration across tightly and loosely coupled heterogeneous computational architectures.

In section 4 results are presented for MiBench benchmarks demonstrating the ability of 3SP to identify significant program acceleration potentials through its automatic binary partitioning algorithm for heterogeneous architectures. Further, the results demonstrate the applicability of 3SP as a design-tool for rapid investigation and visualisation of program acceleration opportunities, sensitivities and transition points through the comprehensive exploration of the design space for possible hardware and software architectures.

7. ACKNOWLEDGEMENT

The support of the UK EPSRC is gratefully acknowledged. Dr O. Mencer, Dr D. Thomas and Dr T. Todman provided valuable contributions.

8. REFERENCES

[1] S.A. Spacey. 3S: Program Instrumentation and Characterisation Framework. *Imperial Technical Paper 2008/1* `http://www.doc.ic.ac.uk/research/technicalreports/2008/DTR08-1.pdf`, 2006.

[2] G. Stitt, F. Vahid. Hardware/Software Partitioning of Software Binaries. *IEEE/ACM International Conference on Computer Aided Design*, pp 164–170, 2002.

[3] R. Lysecky, F. Vahid. A Configurable Logic Architecture for Dynamic Hardware/Software Partitioning. *Design Automation and Test in Europe Conference (DATE)*, pp 480–485, 2004.

[4] G. Sitt, F. Vahid. A Decompilation Approach to partitioning Software for Microprocessor/FPGA Platforms. *Design Automation and test in Europe (DATE)*, pp 396–397, 2005.

[5] K. Atasu, C. Özturan, G. Dündar, O. Mencer, W. Luk. CHIPS: Custom Hardware Instruction Processor Synthesis. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 3, pp. 528–541, 2008.

[6] G. Sitt, F. Vahid. New Decompilation Techniques for Binary-level Co-processor Generation. *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp 547–554, 2005.

[7] R. Lysecky, G. Stitt, F. Vahid. Warp Processors. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, pp 659–681, 2006.

[8] T. Wiangtong, P.Y.K. Cheung, W. Luk. Hardware/Software Codesign: A Systematic Approach Targeting Data-Intensive Applications. *IEEE Signal Processing*, Vol. 22, No. 3, pp 14–22, 2005.

[9] Y.M. Lam, J.G.F. Coutinho, W. Luk, P.H.W. Leong. Integrated Hardware/Software Codesign for Heterogeneous Computing Systems. *In Proceedings of the Southern Conference on Programmable Logic*, pp 217–220, 2008.

[10] S. Sirowy, Y. Wu, S. Lonardi, F. Vahid. Two Level Microprocessor-Accelerator Partitioning. *Design Automation and Test in Europe (DATE)*, pp 313–318, 2007.

[11] G.C. Hunt, M.L. Scott. The Coign Automatic Distributed Partitioning System. *In Proceedings of the 3rd Symposium on Operating Systems Design and Implementation*, New Orleans, Louisiana, pp 45–52, 1999.

[12] A. Kalavade, E.A. Lee. A Global Criticality/Local Phase Driven Algorithm for the Constrained Hardware/Software Partitioning Problem. *International Workshop on Hardware/Software Codesign*, pp 42–48, 1994.

[13] P. Eles, Z. Peng, A. Kuchcinski, A. Doboli. System Level Hardware/Software Partitioning Based on Simulated Annealing and Tabu Search. *Journal on Design Automation for Embedded Systems*, vol. 2, pp 5–32, 1997.

[14] M.W. Hall, J.M. Anderson, S.P. Amarasinghe, B.R. Murphy, S. Liao, E. Bugnion, M.S. Lam. Maximizing Multiprocessor Performance with the SUIF Compiler. *IEEE Computer (A special issue on multiprocessors)*, 1996.

[15] O. Mencer, D.J. Pearce, L.W. Howes, W. Luk. Design Space Exploration with A Stream Compiler. *IEEE International Conference on Field Prog. Tech.*, 2003.

[16] Celoxica Ltd. Handel-C Language Reference Manual. 2004.

[17] N. Nethercote, J. Seward. Valgrind: A Program Supervision Framework. *Proceedings of the 3rd Workshop on Runtime Verification* 2003.

[18] C. Luk *et al.*. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*, 2005.

[19] D.J. Pearce, P.H.J. Kelly, T. Field, U. Harder. GILK: A Dynamic Instrumentation Tool for the Linux Kernel. *Proceedings of the 12th International Conference on Computer Performance Evaluation, Modelling Techniques and Tools 37*, pp 220–226, 2002.

[20] B. Holden. Latency Comparison Between HyperTransport and PCI-Express In Communications Systems. *HyperTransport Consortium*, 2006.

[21] M.R. Guthaus *et al.*. MiBench: A Free, Commercially Representative Embedded Benchmark Suite, *IEEE 4th Annual Workshop on Workload Characterization*, 2001.

[22] Sharpe, W.F. Mutual Fund Performance. *Journal of Business*, pp 119–138, 1966.

[23] W. Wiesemann, D. Kuhn, B. Rustem. Multi-Resource Allocation in Stochastic Project Scheduling. *Annals of Operations Research*, 2008.

[24] S. Sahni, T. Gonzalez, P-complete Approximation Problems. *Journal of the ACM (JACM)*, Vol. 23, pp. 555–565, 1976.