Department of Electrical
and
Computer Systems Engineering

Technical Report
MECSE-10-2007

Estimating the communications cost of application accelerator attached processors through memory address tracing

T. Ramdas and G. Egan

MONASH
UNIVERSITY

Estimating the communications cost of application accelerator attached processors through memory address tracing

Tirath Ramdas
tirath.ramdas@eng.monash.edu.au

Greg Egan
greg.egan@eng.monash.edu.au

Centre for Telecommunications and Information Engineering
Monash University, Melbourne, Australia

Abstract

Application specific processor (ASP) accelerators are an important aspect to overcoming the growing disparity between theoretical peak performance and sustained actual performance for key scientific applications with general purpose systems. A potential stumbling block is that the performance gains of the ASP would be lost due to communications limitations between the host system and the ASP. In many instances, an analytical model of the kernel of the application is available to make strategic decisions regarding what to do on the host and what to do on the ASP, with communications minimisation in mind. Unfortunately, there are instances where such straight-forward analytical models are not available. We propose the use of memory address tracing to derive an estimate of relative communications expense for candidate ASP kernels. To demonstrate the viability of the method, we present two concrete example application kernels: molecular dynamics and sequence similarity detection. We go on to present preliminary findings of the method on a computational quantum chemistry application.

1 Introduction

As programmable logic devices, specifically *Field-Programmable Gate Arrays* (FPGAs), achieve very large capacities, and as systems integrators such as *SRC*, *SGI* and *Cray* appear motivated to provide high performance interconnects to FPGAs (and with high performance commodity interfaces becoming mainstream as well), the viability of attached processor (AP) application accelerators, or application specific processors (ASPs), seems to encompass an ever growing range of applications. A survey of *Smith-Waterman* bioinformatics accelerators was presented in [8], where all the surveyed accelerators were integrated with a host system thru PCI. Similar speedups have been reported for applications such as molecular dynamics [3] and floating-point matrix multiplication [2].

However, failures also abound. For example, an initial attempt at implementing a *BLAST* bioinformatics accelerator [6] was unsuccessful in producing speedup, and the authors attribute this to a communications bottleneck between the host and the accelerator.

FPGA-based accelerators are attractive for obvious economic reasons, and the dramatically reduced "barrier-to-entry" precipitates the exploration of application accelerators which would not have the economic momentum to warrant *Application Specific Integrated Circuits* (ASICs). Having said that, this paper is concerned only with ASP and host system *communications*, and therefore distinctions between the fabric upon which the application accelerator is built – be it FPGA, ASIC, or a mixture of both – will be disregarded.

The *MD-GRAPE* [11] is widely held as being a highly successful application accelerator, where the application is molecular dynamics. A salient point is that the core of the application – i.e. Coulombic force term evaluations – exhibits $O(N^2)$ computation time with $O(N)$ data requirement (where N is the number of atoms in the system under study). This implies that any communications cost will be easily amortised by the very large computational requirements of the application kernel. The same reasoning applies to the previously mentioned *Smith-Waterman* application accelerators: here too computation time is $O(N^2)$ with $O(N)$ data transfer.

We present a method to develop an understanding of the asymptotic computation vs. communication behaviour of a candidate for ASP deployment. Our method is based on instruction and memory tracing. This method allows one to begin to understand the behaviour and requirements of a candidate kernel very early in the project lifecycle, without the need for an analytical understanding of the kernel and the overall application.

The proposed method falls within the domain of hardware/software codesign. Indeed, many hardware/software codesign partitioning schemes enforce communications cost as a constraint. One solution is PACE [4], which is a sub-system within the LYCOS system [5]. This system requires input in the form of a subset of *C* and *VHDL*, which severely limits the applicability of the method to existing applications, such as GAMESS [9] which is written in *Fortran 77*.

LYCOS is not unique in this regard. Systems such as POLIS [1] and many others also require specification to be in the form of either a subset of some language or an entirely new language. Systems such as OCAPI-xl [13] and SystemC [10] employ special object classes to encapsulate multiple system specifications for a given object.

Many existing applications will not be able to directly exploit these systems, and ultimately it may be necessary to port over parts of the application to a new language/specification, which requires an analytical understanding of the application kernel. Unless educated

decisions can be made about where to start the porting activity, this will often be an intractable job. We propose a methodology to aid such decision making.

The method we proposed can be used when an analytical model of the target application does not yet exist, and regardless of what language the application is written in.

The contributions of this paper are:

- A demonstration of the application of trace analysis towards asymptotic computation vs. communication comparison of potential host/ASP partitions for early design space exploration. We propose and define several new metrics in the following section that allow memory tracing to be applied towards estimating the communications cost of candidate host/ASP partitions.
- The proposed method does not rely on an analytical model of the candidate kernel.
- The proposed method does not rely on language specific mechanisms such as object classes, pre-processor directives (e.g. *#pragmas*), existing IP blocks etc. and is therefore a viable early estimation tool for a very wide range of existing applications. The only required program augmentation is the instrumentation of the candidate partition with calls to start and stop tracing which, with the currently prescribed trace extraction tool, is a trivial task.
- The results obtained with this method can be used to suggest partitioning schemes to HW/SW co-design systems that rely on user-driven partitioning advice, such as POLIS [1].
- The proposed method is stimulus based and therefore candidate partitions may be exercised with real-world data.
- An initial investigation into communications requirements of a potential host/ASP partition for a computational quantum chemistry program, i.e. *GAMESS* [9].

This paper presents the proposed methodology, two test cases, and preliminary results for the method on a realistic computation lacking a clear analytical model. Section 2.1 describes the methodology and many of the terms adopted for the rest of the paper. Section 2.2 describes the two test cases, i.e. molecular dynamics and Smith-Waterman. Some limitations of the method are explained in sections 2.3 and 2.4. Section 3 presents our initial findings of the method on a candidate partition for GAMESS. Specific gaps to be addressed with further work are briefly discussed in section 4. Details on the software utilised by this method are provided in section 5, followed by concluding remarks.

2 Memory Trace Analysis

Uhlig and Mudge present a survey of memory trace analysis methods and techniques in [12]. Instruction and memory address tracing is well-entrenched in supercomputing; here we discuss how tracing may be adopted for host/ASP partition evaluation. For details on the software components used for this analysis, see section 5.

2.1 Methodology

The general sequence of steps of this method are as follows:

1. The candidate partition is instrumented with calls to start and stop tracing. This requires that all possible paths in the relevant sections of the code be accounted for. A partition may be adopted along existing subroutine boundaries, and therefore instrumenting the partition would be achieved by simply inserting a call to start tracing before the candidate routine is called, and inserting a call to stop tracing once the routine returns.
2. Selection of stimulus (i.e. input) for the program that exercises the candidate partition. In some cases, this is a simple step as the computational procedure of the candidate partition does not differ according to input, however in some cases there are different computational paths for input that meet different conditions.
3. The program is executed and the instruction trace is generated. Subsequently the memory address trace is extracted.
4. The memory address trace is divided into separate read and write traces, but the sequence of reads and writes must be maintained.
5. The number of *read-before-write* (RbW) samples and *write-after-read* (WaR) samples are counted.

The WaR term used here is familiar in the context of internal processor pipeline data dependencies, but perhaps less so in the context of memory tracing. The meaning of the term WaR as used here is slightly different than the use of the term in the context of data dependencies. The two conditions may be described as follows:

RbW A memory address is referenced *exclusively* for reading, or the *first* reference to the address is for reading, and subsequent references may include writes and reads. This accounts for data transfer from the host to the ASP (*toASP*).

WaR A memory address is referenced *exclusively* for writing, or the *last* reference to the address is for writing, and previous references may include writes and reads. This accounts for data transfer from the ASP to the host (*toHost*).

The above labels are applied to memory addresses, not memory references. A count of the number of RbW samples provides an estimate of the amount of data that needs to be *passed to the routine* – i.e. input data, which we shall call *toASP* – and a count of the number of WaR samples provides an estimate of the amount of data that is generated by the routine to be *passed to other routines* – i.e. output data, which we shall call *toHost*. Memory addresses that fail to meet these two conditions may be considered *internal scratch space*; while counting this could be useful in estimating the amount of local memory that the ASP needs, this matter is outside the scope of this paper.

2.2 Test Cases

We employ two test cases to verify that the method works: Molecular Dynamics long-range Coulombic force term evaluation, and simplified Smith-Waterman sequence similarity score reporting. The kernels exercised here are simplified and well understood analytically, and are therefore suitable for demonstrating the method's abilities and limitations in exposing communications requirements.

The molecular dynamics test case is considered first. The component of the code specifically under study is the evaluation of the long-range Coulombic interactions between all atoms in the molecule; therefore for N atoms there would be N^2 force terms to evaluate.

We instrument the kernel responsible for computing the net force

experienced by an atom due to the inter-atomic charge interaction with all other atoms, i.e.:

$$F_i = \sum_{a=0 \rightarrow N, a \neq i} \frac{Q_i \times Q_a}{r_{i,a}} \quad (1)$$

for all atoms in the system, $i = 0 \rightarrow N$. A random constellation of atoms is generated as a stimulus. The program is then executed, and instruction tracing is performed. After that the generated tracefile is processed, producing the results depicted in figure 1. The results agree with the analytically predicted behaviour.

In plots 1(a), 1(b) and 1(c) N is the number of atoms in the molecule under simulation. Plot 1(a) depicts the computation time scaling of the kernel. Plot 1(b) depicts the memory space scaling of the kernel. Plot 1(c) depicts the RaW and RbW scaling of the kernel.

Note plot 1(d); the instruction scaling is plotted vs the corresponding address scaling, and we define the behaviour exhibited in this plot as *operational intensity*. This plot effectively combines figures 1(a) and 1(b). The more *operationally intense* a kernel is (as opposed to *operand intense*) the more suitable it is for ASP deployment, at least in terms of minimising the impact of communications overhead. Operational intensity provides a convenient means to compare different application kernels with different interpretations of N. A kernel which exhibits a steeper curve, i.e. larger instruction scaling vs. data scaling, would likely be capable of amortising communications expense thru a large "amount" of computation relative to the "amount" of communication. This metric is useful to compare different applications with different interpretations of N.

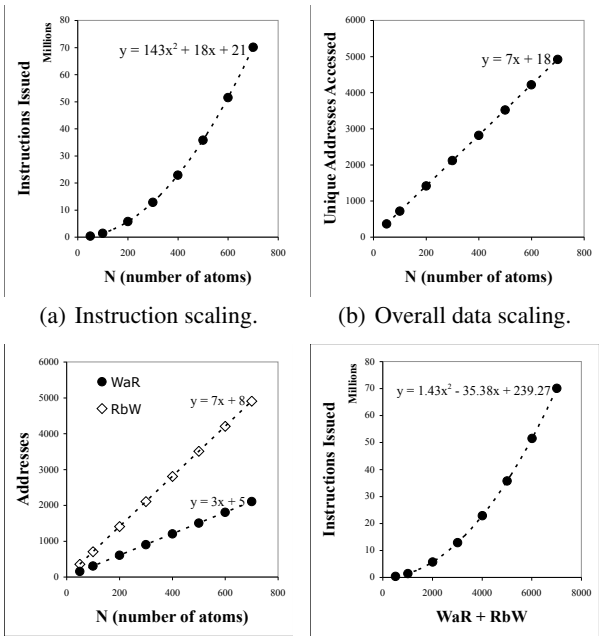
From the plots in figure 1, it is evident that as N increases, the computational cost increases at a rate that dramatically exceeds the required data transfer. Therefore, the cost of data transfer is easily amortised by the computation time.

Figure 1(c) also reveals that the candidate kernel exhibits asymmetric communications requirements, i.e. *toASP* scales as a factor of 7 while *toHost* scales as a factor of 3; therefore 70% of the available communications bandwidth should be allocated for unidirectional data transfer going *toASP*.

We now turn our attention to the second test case application: Smith-Waterman sequence alignment. Two sequences of length N are lined up against a matrix (of dimensions $N \times N$) – it should be pointed out that in practice the two sequences could be of different lengths N and M (and consequently the matrix would have dimensions $N \times M$). Each cell in the matrix is then scored – this step is called *matrix fill*. Figure 2 presents such a matrix, as well as an indication as to how the matrix cells may be populated in parallel – readers interested in a slightly more detailed discussion of this application are referred to [8].

Here we present a slightly contrived variation of a typical Smith-Waterman workflow; typically the matrix fill stage is followed by *traceback*, which reports an alignment sequence. Here, however, we assume that the goal of the algorithm is merely to report the largest single value in the scored matrix, as an indication of the similarity of the two query sequences.

The method is applied to this kernel, and the results are presented in figure 3. Plot 3(a) tells a similar story to the MD case (plot 1(a)) however plot 3(b) reveals the memory space cost of the $O(N^2)$ matrix. However, keep in mind that our main focus here is *communications* expense; the $O(N^2)$ matrix need not be communicated be-



(a) Instruction scaling. (b) Overall data scaling.

(c) RbW and WaR scaling. (d) Operational intensity scaling.

Figure 1. MD instruction and data scaling.

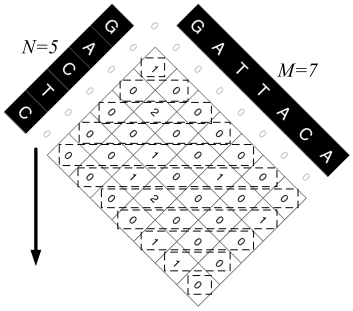


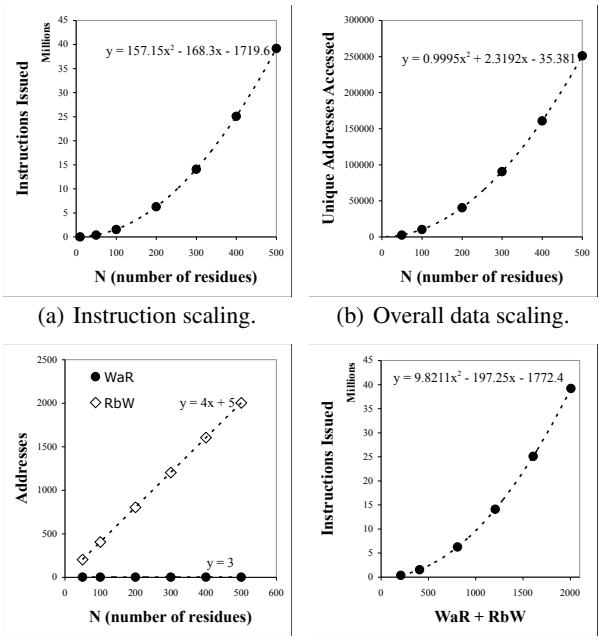
Figure 2. Smith-Waterman scoring matrix parallelisation; this issue is beyond the scope of this paper, see [8] for more information.

tween the host and the ASP, as it is merely *internal scratch-space* on the ASP. In fact, recall that the only data reported by the ASP (i.e. sent *toHost*) should be the single largest cell value – in other words we expect that the *toHost* communications requirements should be *constant*, regardless of N. This is reflected in the $O(1)$ scaling of WaR in plot 3(c). The *toASP* scaling is $O(N)$; the input required are the two query sequences of length N. The operational intensity of the kernel, plot 3(d), tells a similar story to the MD case (plot 1(d)).

2.3 Communicated Scratch-Space

One potential pitfall of this approach is that there is no direct provision for including the cost of communicating *scratch space*. This is poignant because in some situations, it is necessary to pass scratch space from one part of the program to another, and if a partition is adopted that requires the transfer of scratch space, the proposed method must be able to detect this.

Consider a slightly more realistic Smith-Waterman implementation



(a) Instruction scaling. (b) Overall data scaling.

(c) RbW and WaR scaling. (d) Operational intensity scaling.

Figure 3. SW instruction and data scaling. N is the number of residues in the sequences being compared.

than the one discussed thus far: once the scoring matrix is filled, *traceback* must then be performed (see [8] for an explanation of the complete Smith-Waterman algorithm) in order to identify alignments between two query sequences. Generally, traceback entails traversal over the entire scoring matrix. A possible partition is to perform the matrix fill operation on the ASP, and traceback on the host. This would require that the scored matrix be transferred from the ASP to the host, which is clearly far more expensive in terms of ASP-host communications than merely reporting the maximum similarity score. We would expect that this more thorough implementation of SW would exhibit poorer performance – specifically, stronger WaR scaling – than the previously obtained results for "simplified" Smith-Waterman, which is presented in figure 3.

Exposing this expense within the framework of the proposed methodology is problematic because the memory addresses associated with the scored matrix are identified as *scratch space*. The general problem at hand is detecting when scratch space may in fact need to be transferred from the ASP to the host. We shall call this the "communicated *scratch space*" (CSS) problem.

Eliminating the possibility of the CSS problem would, in most cases, be as simple as confirming that the WaR and/or RbW scaling is equivalent to the *unique addresses accessed* scaling.

An easy and conservative approach to resolving the CSS problem would be to abandon the use of the WaR and RbW metrics altogether, and adopt the *unique addresses accessed* (UqA) metric (figures 1(b) and 3(b)) as a communications requirement scaling estimate. This approach has the disadvantage that "genuine" *scratch space* that is not accessed outside the candidate partition would be erroneously included in the communications requirements estimate. Figure 4 compares the functional intensity of SW when UqA and RbW+WaR are taken as data metrics. The UqA plot is indicative of

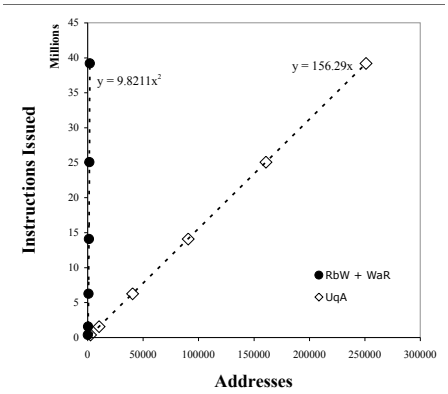


Figure 4. Operational intensity of SW, adopting UqA as a data communications estimate, vs. RbW+WaR as a data communications estimate. The UqA case corresponds to transferring the $O(N^2)$ matrix *toHost*, while the RbW+WaR case corresponds to transferring the maximum cell value *toHost*.

the SW kernel which transfers the entire $O(N^2)$ matrix *toHost*, and clearly exhibits much lower operational intensity than the originally discussed (albeit not very useful) SW implementation which only transferred the maximum matrix cell score *toHost*, i.e. $O(1)$. [1]

A more sophisticated (and less conservative) solution to this problem may require additional tracing on the program with additional instrumentation of partitions of interest. Thus far, only tracing of partitions considered for ASP deployment has been mandated. Accurate CSS resolution may require additional tracing of partitions *not* considered for ASP deployment. The specific goal is to identify if any of the previously identified *scratch-space* within the ASP candidate partition is accessed as RbW by subsequent computations. Specifically looking for RbW accesses to addresses marked as *scratch space* would eliminate the possibility that the range of addresses has been recycled for unrelated scratch-space. Demonstrating this will be left for future work.

2.4 Other Pitfalls and Limitations

Besides the CSS problem, there are a few other limitations and potential pitfalls with this method:

A key assumption made by this method is that any memory locations accessed by the candidate partition are modified only by the candidate partition. If a portion of memory is set up to be shared between multiple threads of execution, with the candidate partition contained within thread A, and thread B modified the shared data, then this method would no longer be applicable.

Complete instruction traces can consume massive amounts of storage; currently, seconds of CPU time tracing could result in trace file sizes in the order of hundreds of megabytes to gigabytes. Furthermore, the overhead of tracing in terms of wall-clock time can also be massive. Having said that, trace analysis is very valuable in understanding computational kernels, and such traces would be

[1] Ideally, the ASP would also perform traceback, and therefore *toHost* communications would scale as $O(N)$ (not shown in this paper), which is clearly poorer than $O(1)$, however this is unavoidable if a complete SW operation, i.e. including alignment reporting, is desired.

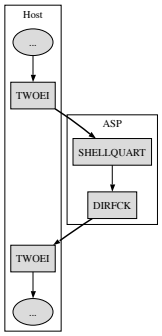


Figure 5. A potential host/ASP partitioning for GAMESS.

desirable beyond the scope of just this method. In fact, trace files may already exist, and this method may be applied on these traces.

The robustness of this method relies greatly on the significance of each and every memory access made by the program. If data accesses are explicitly made in a manner that do not reflect the actual computational requirements of the program, for example as a means to pre-cache data, this could distort the findings of this method. However, modern processors now have dedicated cache-specific instructions, which should minimise this specific problem.

3 Applied to GAMESS

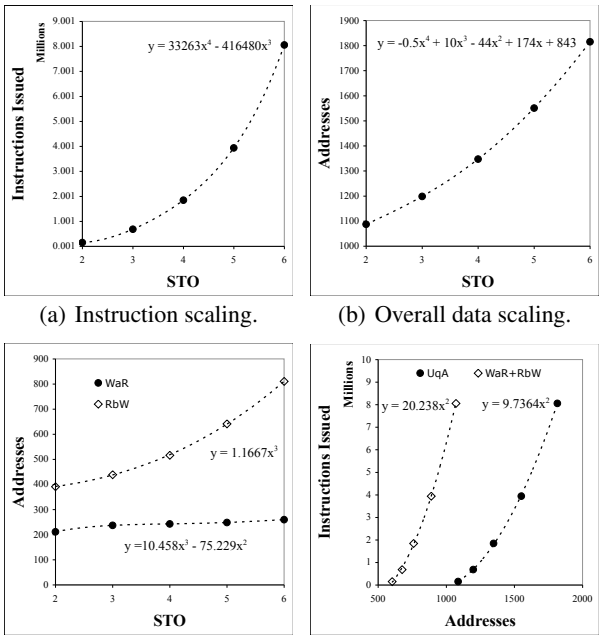
The *General Atomic and Molecular Electronic Structure System* (GAMESS) has a long history dating back to the 1970s, and is a very popular *FORTRAN 77* numerical package in the quantum chemistry domain. A potential host/ASP partitioning scheme, which we shall call the *GAMESS SD-partition*, is depicted in figure 5. The subroutines *Shellquart* and *Dirfck* (and all other subroutines contained therein) will be implemented on the ASP board.

As with the previous MD and SW test cases, a variable N must be defined for computational quantum chemistry. The most suitable choice of N for the candidate partition is the number of *basis functions* involved in the computation. A basis function may be thought of as a collection of Gaussian primitive functions, and the number of basis functions in a computation is determined by the number of atoms in a system, the kind of atoms, the desired accuracy etc. Here, we fix the number of atoms (in fact we use the same molecule for all tests – $C_{10}H_8$) and vary the level of accuracy required. The details are omitted here; suffice to say, with regards to the plots in figure 6 the term STO [2] is effectively the N, and a larger value of STO results in a larger number of basis functions. For those familiar with these computations, the *Rys quadrature* approach was used with *Direct-SCF*. More details will be presented in a future communication.

Note the *very* strong computation time scaling in plot 6(a). Although the memory space scaling of the partition in plot 6(b) is also quite strong, it is not as strong as the time scaling. This is also the case with RbW and WaR scaling in plot 6(c). Overall, the partition exhibits a large degree of operational intensity in plot 6(d), regardless of whether UqA or RbW+WaR is adopted as data scaling.

Comparing the operational intensity of the GAMESS SD-partition

[2] Actually, it was the *NGAUSS* variable that was changed in the GAMESS input file, while the *BASIS* variable was fixed at STO.



(a) Instruction scaling. (b) Overall data scaling.

(c) RbW and WaR scaling. (d) Operational intensity scaling.

Figure 6. GAMESS SD-partition instruction and data scaling.

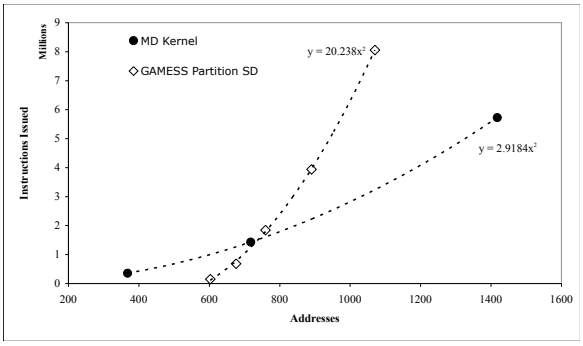


Figure 7. Operational intensity: MD kernel vs GAMESS Shellquart-Dirfck partition. Conservative (i.e. UqA) data scaling was assumed for the GAMESS SD-partition.

vs. the MD kernel in figure 7, we can see clearly that the GAMESS SD-partition exhibits a greater operational intensity scaling. This is quite encouraging with regards to the possible real-world wall-clock speedup that an ASP may yield for the target application, assuming of course that it would be possible to construct a high-performance ASP (which is an especially critical point if FPGAs are favoured over ASICs) for the candidate partition. Clearly, this is a critical point, however it is a point that is outside the scope of this paper.

4 Further Work

The CSS problem described in section 2.3 should be addressed. A concrete example of how to overcome the problem by performing additional tracing as was prescribed in section 2.3 should be developed.

Thus far, a transactional *lumped* data transfer model has been as-

sumed. There has been no accounting for the possibility of staggering data transfers over the timeline of the computation. The memory traces produced retain timing information, i.e. the time between data accesses (in clock cycles) is recorded. Therefore, it may be possible to infer the time between consecutive RbW and WaR accesses, which could be used to depict a gradual data transfer continuum rather than a single *toASP* transfer and a single *toHost* transfer.

The robustness of this method is affected by, among other things, compiler optimisations on the profiled kernel. It would be prudent to consider the impact various compiler optimisations may have on a candidate kernel that would distort the findings obtained with this method.

Specifically with regards to the application of the method to GAMESS, the obvious avenue for future work is the evaluation of different partitions. While the existing partition exhibits favourable communications scaling, implementing the entire Shellquart and Dirfck sub-graph may not be practical in terms of resources and the level of effort required to design the ASP.

We rely on a general trace extraction tool (see section 5). However, we are only interested in load/store instructions, and in fact in the post-trace processing we simply disregard non-load/store instructions. All instructions would need to be counted, but only load/store instruction words need to be recorded. Therefore, if we were to develop a more specialised trace extraction tool we would be able to decrease the storage requirements of the trace files. With one example trace, this would have resulted in savings of over 55%.

5 Software Stack

The software stack used to produce the data presented in this paper include *amber* and *acid*, which are part of Apple Computer's *Computer Hardware Understanding Developer* (CHUD) toolkit. The test programs (i.e. the MD kernel, the SW kernel, and GAMESS) were instrumented with a supervisor assembly instruction that, when executed in normal user-mode, would result in an *illegal instruction* exception (i.e. *SIGILL*) and therefore cause the program to terminate abnormally. However, when the instrumented program is run with *amber*, the *SIGILL* is trapped and triggers the start of trace collection. The next *SIGILL* would stop trace collection. The tracefiles are then pre-processed with *acid* to extract data addresses from the tracefile. This output is then processed with a utility developed to extract the WaR and RbW metrics. This utility may be obtained from [7].

It should be noted that while we present our methodology on a trace analysis framework, the methodology itself is immediately portable to a simulation framework. An existing simulator such as SimpleScalar could be instrumented to count the number of UqA, RbW, WaR, and instructions executed, and then produce the plots that we prescribed.

6 Conclusion

We have presented a method for estimating the asymptotic computation vs. communication behaviour of any arbitrary computational kernel through instruction and memory address tracing, for the purpose of performing an early evaluation of the viability of deploying the kernel on an ASP. Through the use of two concrete computational kernels – molecular dynamics Coulombic force term evaluation, and the Smith-Waterman sequence alignment algorithm – we have demonstrated the method's capacity for exposing analytically expected behaviour. Furthermore, utilising the method, we have provided initial insight into the feasibility of an ASP for a computational quantum chemistry application.

The main strength of this method lies in the fact that it allows one to attack a problem with a stimulus driven approach before an analytical model is developed and without reliance on particular specification languages. It is by no means proposed that the method is a replacement for other analytical approaches. It's main value is as a tool to empirically estimate the ability of a prospective ASP kernel to operate independently of the host, which translates to less reliance on a high performance host-ASP interconnect. Kernels that pass this initial test will then be tackled with more formal methods, and perhaps expressed in proper HW/SW co-design specification languages.

7 References

[1] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, and B. Tabbara. *Hardware-Software Co-Design of Embedded Systems: The Polis Approach*. Kluwer Academic Publishers, 1997.

[2] Y. Dou, S. Vassiliadis, G. K. Kuzmanov, and G. N. Gaydadjiev. 64-bit floating-point fpga matrix multiplication. In *FPGA '05*, pages 86–95, New York, NY, USA, 2005. ACM Press.

[3] Y. Gu, T. V. Court, and M. C. Herbordt. Accelerating molecular dynamics simulations with configurable circuits. In *Proceedings of FPL 2005*, pages 475–480, 2005.

[4] P. V. Knudsen and J. Madsen. Pace: A dynamic programming algorithm for hardware/software partitioning. In *CODES '96*, page 85, Washington, DC, USA, 1996. IEEE Computer Society.

[5] J. Madsen, J. Grode, P. Knudsen, M. Petersen, and A. Haxthausen. Lycos: The lyngby cosynthesis system, 1997.

[6] K. Muriki, K. D. Underwood, and R. Sass. Rc-blast: Towards a portable, cost-effective open source hardware implementation. In *Proceedings of IPDPS*, 2005.

[7] T. Ramdas. process_trace. http://users.monash.edu.au/~tramdas/process_trace, 2006.

[8] T. Ramdas and G. Egan. A survey of fpgas for acceleration of high performance computing and their application to computational molecular biology. In *Proceedings of IEEE TENCON*, 2005.

[9] M. W. Schmidt, K. K. Baldridge, J. A. Boatz, S. T. Elbert, M. S. Gordon, J. H. Jensen, S. Koseki, N. Matsunaga, K. A. Nguyen, S. Su, theresa L. Windus, M. Dupuis, and J. A. M. Jr. General atomic and molecular electronic structure system. *Journal of Computational Chemistry*, 14(11):1347–1363, 1993.

[10] S. Swan. An Introduction to System Level Modeling in SystemC 2.0, 2001.

[11] M. Taiji, T. Narumi, Y. Ohno, N. Futatsugi, A. Suenaga, N. Takada, and A. Konagaya. Protein explorer: A petaflops special-purpose computer system for molecular dynamics simulation. In *Proceedings of Supercomputing 2003*, 2003.

[12] R. A. Uhlig and T. N. Mudge. Trace-driven memory simulation: a survey. *ACM Comput. Surv.*, 29(2):128–170, 1997.

[13] G. Vanmeerbeeck, P. Schaumont, S. Vernalde, M. Engels, and I. Bolsens. Hardware/software partitioning of embedded system in ocapi-xl. In *CODES '01*, pages 30–35, New York, NY, USA, 2001. ACM Press.