The Role of Specialized Processors in the Future of HPC – A Preliminary Study on the Implementation and Exploitation of Specialized Linear Algebra Processors (SLAPs)

JUAN GONZALEZ, SANTIAGO FONSECA, RAFAEL NUNEZ, Accelogic, LLC

The advent of modern heterogeneous processing technologies (e.g., multicore CPUs, GPUs, FPGAs) poses an opportunity for significantly increasing the performance of supercomputer codes. These technologies are enabling massive computational parallelism as well as revolutionary approaches to memory-access speedup that could potentially reduce computation times of complete applications in the order of thousands. Exploiting these technologies in HPC environments is not trivial –it requires appropriately balancing processing and data transfer performance, as well as providing programming tools able to keep development costs comparable to those of conventional software environments. We introduce the concept of Specialized Linear Algebra Processor (SLAP) as a low-cost processor with the ability to compute certain specialized numerical kernels at speeds comparable to (or higher than) those achieved by general-purpose supercomputers. SLAPs can be placed strategically inside a supercomputer network to enhance its performance for certain applications, with relatively low investments in additional hardware or development costs. For example, codes in the FPGA-accelerated LAPACKrc™ library, could be used within a supercomputer enhanced with one or a handful of commercial-off-the-shelf FPGA systems, to inject substantial acceleration into certain applications. To illustrate the potential of SLAPs, we present a demonstration of the NAS CG benchmark that makes use of the Krylov solver component of LAPACKrc™ in a state-of-the-art FPGA system. Results indicate that for certain NAS benchmarks, just a single SLAP could outperform an entire supercomputer, which emphasizes the importance of Specialized Processors in the future of HPC.

Categories and Subject Descriptors: **F.2.1 [Numerical Algorithms and Problems]**

General Terms: Linear Algebra, Algorithms, Design, Theory

Additional Key Words and Phrases: Specialized processors, GPU, FPGA, High Performance Computing, Distributed Algorithms, NAS Benchmarks,

ACM Reference Format:

Juan Gonzalez, 2011. The Role of Specialized Processors in the Future of HPC – A Preliminary Study on the Implementation and Exploitation of Specialized Linear Algebra Processors (SLAPs).

1. INTRODUCTION

High-Performance Computing (HPC) applications are beginning to embrace a new tool in their constant search for increased performance: Specialized Processors. We define a "Specialized Processor" as a piece of hardware (made of either one or more computational units) in conjunction with *highly specialized software or firmware* that enables the hardware to compute specific application kernels (1) *with substantial ease of use*, (2) *at very high speeds*, and (3) *at very low costs per FLOP*. Although it is not our intention to provide a formal black-and-white definition, for the purposes of this discussion it is relatively straightforward to determine when a combination of hardware and software/firmware can be considered a specialized processor. The key

This work is based on research supported by the U.S. Air Force under grant FA9550-09-C-0052, and NASA under grant NNX10RA04C.

Author's address: J. Gonzalez, Accelogic, LLC; 1633 Bonaventure Blvd, Weston, FL 33326.

is for the attributes numbered above holding true for the particular kernel(s) of specialization. This conceptualization is summarized in Table 1.

Table 1. Attributes expected from a computational system for it to be considered a *Specialized Processor*, as defined in this paper.

Attribute	Description
Specialization	A combination of special hardware infrastructure, software, and algorithms, makes the system better suited to compute the kernels of specialization at higher speeds and with increased efficiency. There is likely some level of customization and/or of substantial investment (either in hardware infrastructure or in algorithm R&D) in order to attain increased performance.
Speed	The speed at which the system computes the kernels of specialization is much higher than the speed that can be achieved by comparable traditional systems. In some cases, such high level of speed could prevent the scaling to multiple specialized systems due to inexorable bottlenecks in data movement. In these cases, one specialized processor would be faster than two or more.
Ease of Use	Even though the implementation of the specialized software and algorithms can be extremely complex and take years to be produced, the system is designed to be acquired and used in a straightforward manner. There is no requirement of depth of understanding of the implementation in order for the user to be able to use and exploit the high speed and efficiency of the system.
Cost Advantage	When operating at high speeds, the cost of the "specialized FLOP" rendered by the system is much lower than that of a traditional system able to operate at similar speeds.

Note that we are not proposing particular requirements as to the general programmability of the system, or the ease of the programmability or development, if any. The only requirement is for the system to be easy to use by an unsophisticated user, and for it to perform well in its field of specialization.

Examples of Specialized Processors include many efforts on high-performance numerical libraries and custom designs developed for accelerators or clusters of accelerators such as GPUs and FPGAs (e.g., [1][2][3][4][5]), specialized ASICs, and high-end custom systems such as the Anton molecular dynamics special-purpose machine [16].

Unlike the multi-million dollar Anton machine, which has the ability to replace and outperform the fastest general-purpose supercomputers in its own field of specialization, we argue that lower cost "kernel oriented" Specialized Processors have the ability to complement the operation of a general-purpose supercomputer and help it break some of its bottlenecks, instead of competing against it.

Given that a vast amount of the most important HPC applications today do not scale well on general-purpose supercomputers, and given that in many cases, bottlenecks and scaling walls can be tracked down to commonly known ubiquitous computational kernels, a supercomputer could be easily enhanced with strategically located Specialized Processors able to render "specialized kernel FLOPs" at very high speeds and at much lower costs.[2]

We argue that investing in researching, developing, and ultimately commercializing highly-efficient kernel-oriented Specialized Processors for the most common HPC kernel classes makes not only a lot of financial sense, but will be potentially a key enabler of the evolution of HPC in the coming decade –an enabler perhaps more meaningful and of more impactful implications in real life than the often meaningless quest for blind LINPACK exascale performance.

[2] And, if properly designed, at much lower joule/FLOP counts as well.

An opportunity exists today for developing very-low-cost Specialized Processors for a variety of important computational kernels within a supercomputer network. This is a path similar to the one already walked by CPUs years ago: as transistors and digital real estate became cheaper, it became cost-effective to dedicate custom chip area for built-in specialized high-performance functions that would be used rarely, but with very high speed (e.g., trigonometric functions).

We present in this paper a preliminary study on the implementation and exploitation of a Specialized Processor for the acceleration of linear algebra computations. We call such processor a Specialized Linear Algebra Processor, or "SLAP." In Section 2, we introduce the concept of SLAP, and describe our early developments on this technology. In Section 3, we introduce a SLAP-based sparse iterative linear algebra solver, and in Section 4, we compare the performance of this SLAP against current supercomputing technologies, using the well-known NAS CG benchmarks. We close the paper with a few conclusions and remarks in Section 5. Given the ubiquity of sparse linear algebra in HPC, and the high volume of demand for computation of linear algebra kernels, we argue that it makes sense to invest in SLAP research programs that push on the efficiency of SLAPs further through specialization of both infrastructure and algorithms.

2. SPECIALIZED LINEAR ALGEBRA PROCESSORS (SLAPs)

Encompassing several of the seven dwarfs of High Performance Computing (HPC) [6], linear algebra emerges repeatedly in areas such as aerodynamics, molecular physics, nanoscale science, climate modeling, nuclear physics, astrophysics, combustion, and medical imaging, among others. This ubiquity is a strong force behind mankind's constant search for faster and more efficient computational methods for linear algebra and matrix computations. Based on LAPACKrc™, a successful FPGA/GPU-based acceleration technology for linear algebra, our numerical solutions address this need. Next, we briefly introduce LAPACKrc™ and the concept of SLAPs.

2.1. LAPACKrc™

LAPACKrc™ is a groundbreaking family of linear algebra solvers that increase computational speed using hybrid CPU/GPU/FPGA computing systems. LAPACKrc™ incorporates the traditional linear algebra functionality of libraries like LAPACK, ScaLAPACK, PETSc, and MUMPS, and adds sophisticated general direct and iterative sparse solvers that are at least 100 times faster than legacy solvers. The LAPACKrc™ library is based on LAPACK and ScaLAPACK, the de-facto dense linear algebra libraries for high-performance computing. LAPACKrc™ also contains functionalities for the acceleration of sparse solvers using both direct and iterative methods. Early prototypes of LAPACKrc™ have demonstrated up to 150x speedup factors for the set of linear equation and least-squares components of the library [7][8].

2.2. Specialized Linear Algebra Processors

We introduce the concept of Specialized Linear Algebra Processor (SLAP) as a low-cost Specialized Processor with the ability to compute certain specialized linear algebra kernels at speeds comparable to (or higher than) those achieved by general-purpose supercomputers. SLAPs can be placed strategically inside a supercomputer network to enhance its performance for certain applications, with relatively low investments in additional hardware or development costs. A SLAP could be a custom ASIC specialized in certain recurrent linear algebra computations, or it could be a commercial-off-the-shelf FPGA or GPU accelerator (or a small cluster of these),

equipped with highly-optimized specialized linear algebra solvers that would be easily "callable" from the supercomputer network.

Figure 1 shows, at a conceptual level, the integration of SLAPs into supercomputing networks. When embedded, for example, into a Commercial-Off-The-Shelf (COTS) FPGA system connected to a supercomputer network, LAPACKrc™ transforms the COTS system into an extremely powerful Specialized Linear Algebra Processor. The main characteristics of a SLAP are:

- A SLAP could replace hundreds to thousands of supercomputer processors for certain classes of matrix computations.
- One single SLAP can be faster than certain supercomputers for computations that are heavy on linear algebra.
- SLAPs feature from tens to thousands of times reduction in costs (both in system acquisition as well as on its maintenance and operation).
- SLAPs can be integrated seamlessly into existing supercomputing networks.

It is worth noting that when the concept of SLAP is implemented with programmable technologies such as FPGAs or GPUs, this allows for the configuration of different specialized numerical cores without the need for additional investment in hardware – a strong feature that makes these technologies attractive in the short term.

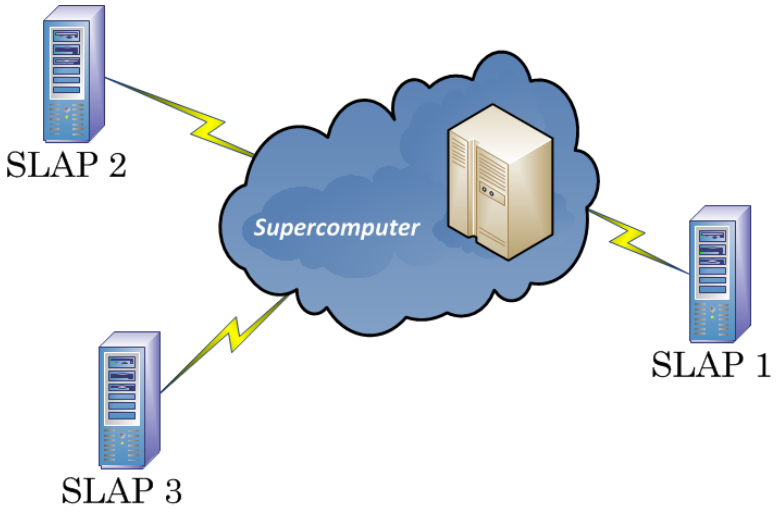

Figure 1. Low-cost Specialized Linear Algebra Processors (SLAPs) could be integrated strategically into key spots of an existing supercomputing network, thus enhancing the supercomputer's ability to reduce time-to-solution for certain linear-algebra-intensive applications.

3. SLAP FOR THE CONJUGATE GRADIENT ALGORITHM

Given the importance that the solution of systems of linear equations has for scientific and engineering applications, the number of techniques available to solve this kind of problem is large, and multiple variants of solvers exist according to the characteristics of the problem to be solved. Solvers are usually classified into two groups: direct and iterative. Direct methods are preferred for problems that involve dense matrices or small and medium-sized sparse matrices. When the matrices are sparse and very large, iterative methods are preferred.

Of the iterative solvers, conjugate gradient (CG) is extensively used in applications today. Reasons include its fast convergence when the model of the system is suited for the use of this method. CG solvers are available in most of the

high-end libraries for numerical computation, including Intel Math Kernel Library [9] and IMSL numerical library [10]. A detailed description of the CG algorithm can be found in [11]. Because of its relevance for large-scale applications, this is the first iterative solver that we extend into the SLAP framework.

The CG method is suitable for solving any linear system $Ax=b$, where the coefficient matrix A is both symmetric, and positive definite. Roughly speaking, CG is a minimization procedure for the quadratic-system

$$\Phi(x) = \frac{1}{2}x^T Ax - x^T b$$

.

A convenient formulation of the CG method for parallelization using a specialized processor is based on repeating the following iteration until convergence is achieved:

$$\begin{aligned} q_i &= Ap_{i-1} \\ \alpha_i &= \frac{\rho_{i-1}}{p_{i-1}^T q_i} \\ x_i &= x_{i-1} - \alpha_i p_{i-1} \\ r_i &= r_{i-1} - \alpha_i q_i \\ \rho_i &= r_i^T r_i \\ \beta_i &= \frac{\rho_i}{\rho_{i-1}} \\ p_i &= r_i + \beta_i p_{i-1} \end{aligned}$$

Here, upper case letters represent matrices, lower case letters represent vectors, and greek letters represent scalars; A is the matrix of coefficients, b is the vector of independent terms, x_i is the solution of the linear system at iteration i, and r_i is a computed equivalent of the vector of residuals $r_i = b - Ax_i$ at iteration i.

The computational bottlenecks of the CG method are the matrix vector multiply (MVM) that computes $q=Ap$, along with the dot product and "axpy" operations. These are thus the computational components subject to receive most of the benefits that specialized computing cores can offer. An increase in the degree of parallelism in these units directly reduces the time per iteration, which is ultimately translated into a faster linear equation solver. Contrary to vector and pipelined processors, which limit the number of parallel operations that can be done every clock cycle, the FPGA specialized computing designs of LAPACKrc™ allow the exploitation of massively parallel units, providing a tool able to compute multiple elements of the MVM output in a single clock cycle.

Figure 2 is a diagram of a top-level architecture for our specialized FPGA-based CG core. The system in **Figure 2** implements all of the operations of a single iteration in the CG algorithm. This system continuously returns the output values x, r, □, and p, which are then fed back to the system for the computations of the next iteration (the initial values are fed to the system by the user at the beginning of the iterative process).

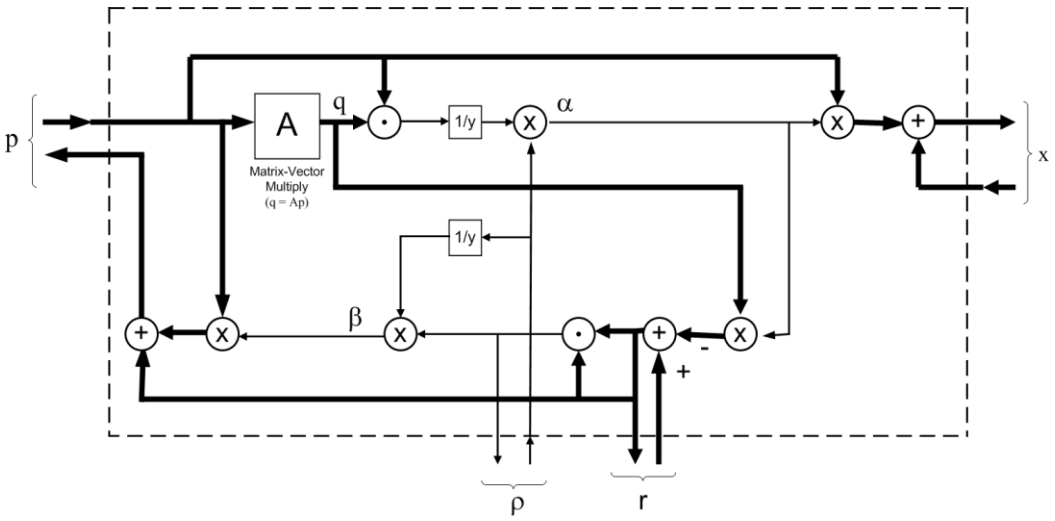

Figure 2. Top-level architecture of a CG solver. The matrix-vector multiply and dot products are the main bottlenecks, which require most of the optimizations in the SLAP solution.

Accelerated computing units, together with novel data-transfer techniques that minimize communication and memory access delays, are fundamental for providing high-performance CG solvers. For relatively small problems, this architecture suffices for accelerating computations. However, large-scale problems do not necessarily take advantage of this solution. As it will be shown in Section 4 below, simply adding computing cores does not translate into increased performance. Memory latency and data transfer must be optimized. Alternatives for solving this issue include recently introduced communication avoidance algorithms [12], as well as new methods that may be incorporated, such as data compression schemes for numerical information and optimized scheduling algorithms.

4. PERFORMANCE ANALYSIS

In this section we describe our preliminary results on the performance of the CG SLAP introduced above. To facilitate a fair comparison with currently existing computing systems, we selected the CG kernel from the NASA NAS Parallel Benchmarks (NPB) [13]. These benchmarks are commonly used to compare the performance of different HPC systems. The NAS CG kernel is defined by a particular formulation of the so-called *inverse power method*, which finds the largest eigenvalue of a symmetric positive definite sparse matrix via repeated iterations of the CG algorithm. Although not an application benchmark itself, the NAS CG kernel gives insight into the performance of a computing system when solving a large system of sparse linear equations via an iterative method – a computational kernel of paramount importance in HPC.

Different problem sizes of the NAS CG kernel are summarized in **Table 2**. In this table, *n* denotes the order of the matrix, *niter* the number of outer iterations in the power method, NONZER the nominal number of non-zeros per row, and □ the so-called *eigenvalue shift* parameter used for the inverse power method.

Table 2. Input parameters for the CG kernel in NPB.

Size	n	niter	NONZER	□
Sample	1,400	15	7	10
Class W	7,000	15	8	12
Class A	14,000	15	11	20

Class B	75,000	75	13	60
Class C	150,000	75	15	110
Class D	1,500,000	100	21	500

We begin the performance study of the NAS CG kernel by analyzing the behavior in single-processor systems. **Figure 3** shows the performance of the kernel (measured in MFLOP/s) when running on a single core of an Intel i7 processor (Westmere) with 4GB RAM running on Windows 7. It can be seen that, as the problem size increases, the rate of execution decreases. An explanation for this loss of efficiency is that as the problem size increases, the use of the cache memory becomes less efficient, and computations stagnate while problem data is retrieved from RAM. This behavior is also present (with the "Sample" Class being an exception) when multiple cores and threads are enabled in the same processor. This is shown in **Figure 4**. By examining this figure, we can also notice that increasing the number of threads does not necessarily mean that the performance will increase (in the Intel i7 processor, performance peaks when the number of threads is equal to the number of cores, but this might not be the case for every numerical kernel). Once the optimum number of threads is reached, the performance begins to degrade. The saturation point depends on the ratio between data transfer and processing operations, as well as on machine specifications such as the cache sizes, data transfer rates, and pipeline sizes.

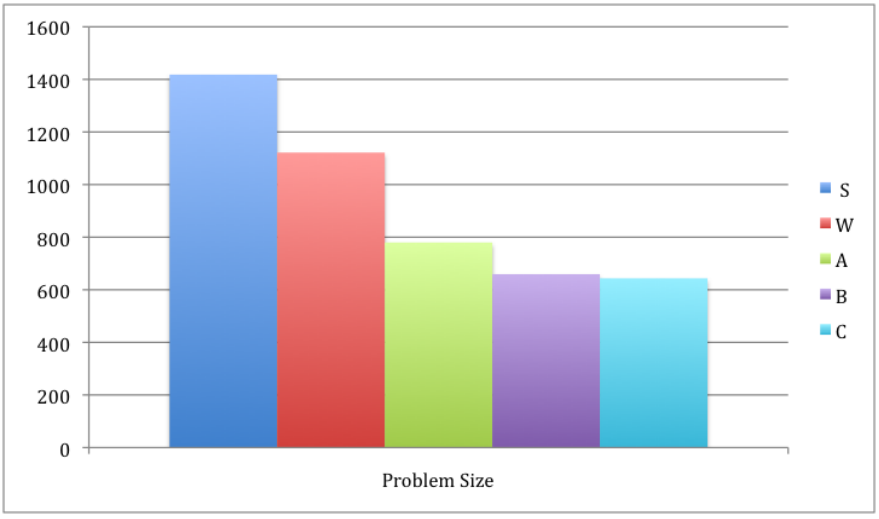

Figure 3. Rate of execution (in MFLOP/s) on a single core of an Intel i7 processor (Westmere, 4GB RAM) as a function of the problem size in the CG kernel of the NAS Parallel Benchmarks (Benchmark by Accelogic, 2011).

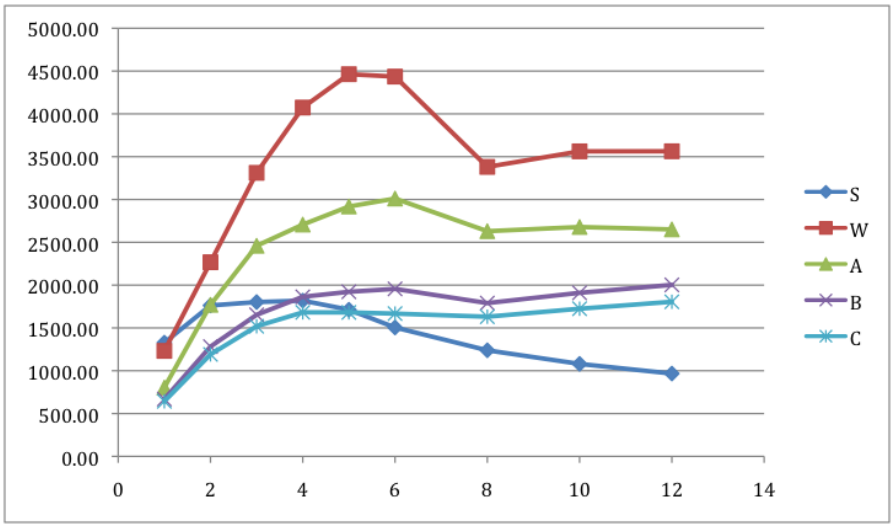

Figure 4. Rate of execution (in MFLOP/s) on an Intel i7 processor (Westmere, 4GB RAM) as a function of the number of threads in the CG kernel of the NAS Parallel Benchmarks (Benchmark by Accelogic, 2011).

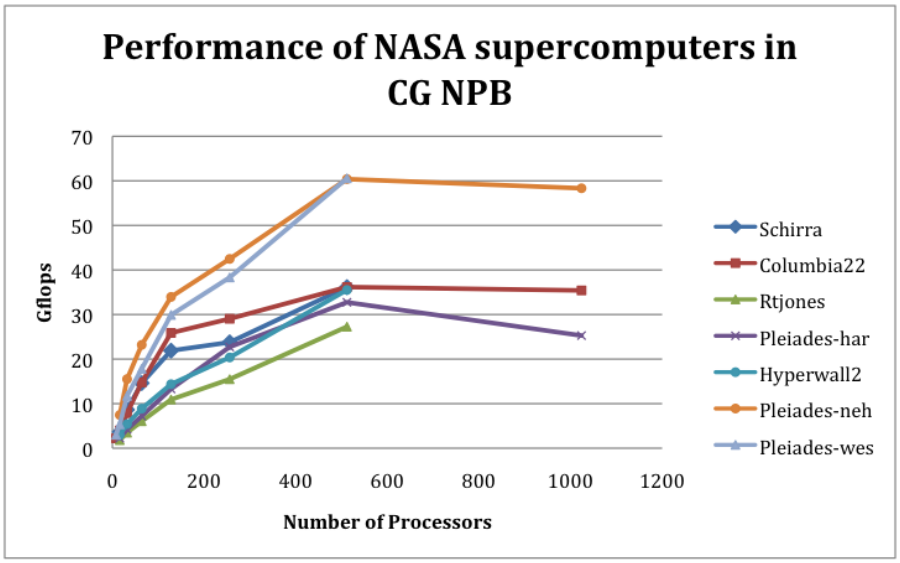

Figure 5. Rate of execution as a function of the number of processors for different supercomputing systems when running the CG NPB Class C Benchmark (Benchmark data provided by NASA NPB Group to Accelogic, 2011).

Performance degradation with multiple cores is not exclusive of single-processor systems. This behavior gets magnified when we work with multi-processor systems. **Figure 5** shows the rate of execution of different supercomputer systems when executing the NAS CG (MPI) Class C kernel (Appendix A details system configurations for these supercomputers).[3] As it can be seen in the figure, the CG benchmark has scaling issues at 256 processors, where most of the systems fall below

[3] We had difficulty obtaining performance data for supercomputers with the Class D problem, but we expect Class D performance to be comparatively consistent with that of Class C.

50% efficiency. The addition of more processors will not result in significant speedup, and could even cause degradation in performance.

Both the single and multi-processor scenarios described above set the performance mark that should be surpassed by a SLAP. In general, SLAPs are vulnerable to some of the limitations of multiprocessor systems, such as the amount of local memory available and communication bandwidths and latencies. However, given that SLAPs can be designed with custom features, they can adjust to the requirements of the target algorithms in order to deliver maximal performance. For example, if necessary, a low-cost SLAP based on FPGA technology can package massive amounts of computing cores in a reduced footprint (e.g., thousands of cores in a single chip), or it can incorporate customized memory streaming architectures for maximal memory access speed, without any of the limitations of a cache-based architecture. Likewise, if custom-designed to be able to handle extremely large matrices, it could accommodate massive amounts of memory at relatively low cost.

Figure 6 shows the performance of the NPB CG kernel using our SLAP, and compares it against a single-processor (Intel i7 –Westmere) solution. The figure illustrates the performance of a kernel prototype running on a heterogeneous system featuring a Dini DNV6_F2PCIe board [14]. This board is populated with two Xilinx Virtex 6 LX760 FPGAs, and 8 GB DDR3 RAM. The FPGA board is hosted on a dual Quad-Core Intel Xeon server and 16GB DDR2 RAM running on Linux Mint 7. **Figure 6** also shows the estimated performance of our SLAP in a higher-end heterogeneous processor, the Convey HC-1 system [15], using our current technology as well as using future improvements (based on theoretical models). It can be seen that the current technology can provide more than 24 GFLOP/s. Moreover, with SLAP optimizations currently under design (see Section 5 below), the current performance can increase to about 81 GFLOP/s, which outperforms all of NASA supercomputers reported in **Figure 5**.

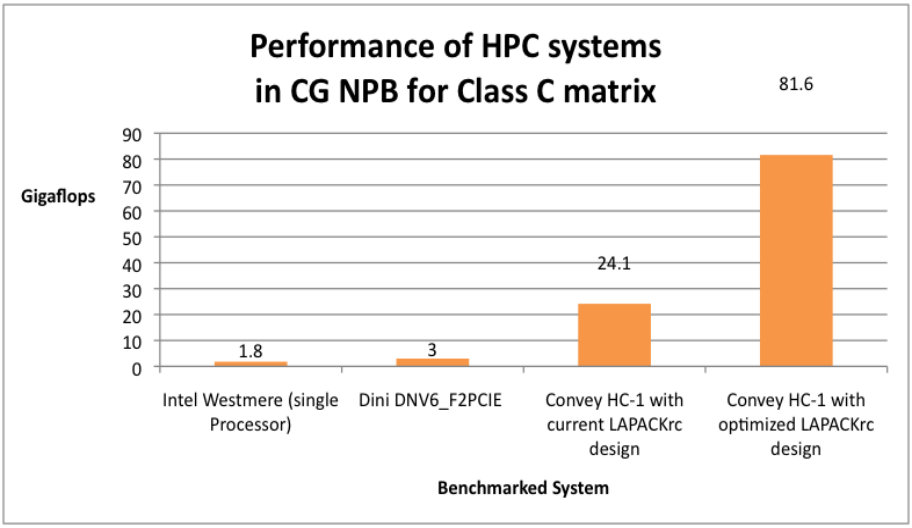

Figure 6. Rate of execution (in GFLOP/s) of NAS CG Class C for the best Intel Westmere configuration and different commercial FPGA systems running CG as an LAPACKrc™ -enabled SLAP (Benchmark by Accelogic, 2011).

Figure 7 compares the performance of the different LAPACKrc™ -enabled CG SLAPs against traditional multiprocessor solutions. In particular, we use the NASA Pleiades-wes supercomputer (the fastest NASA machine for the CG benchmark) as the reference for the benchmark. For the CG NPB, this processor peaks at 60.5

GFLOP/s.[4] The figure shows the number of Pleiades-wes processors that would be necessary to match the speed performance of each SLAP. We note from the figure that matching the performance of the current CG SLAP technology in a high-end FPGA system such as the Convey HC-1, would require more than 100 Pleiades processors –the cost of Convey HC-1 would be significantly less than the cost of 100 processors, and the space and maintenance required would be also significantly reduced. Introducing the optimizations planned for LAPACKrc™ in the near future, would make the Convey HC-1 system perform better (for the CG NPB) than the whole Pleiades-wes supercomputer (hence the "unmatched" mark in the figure).

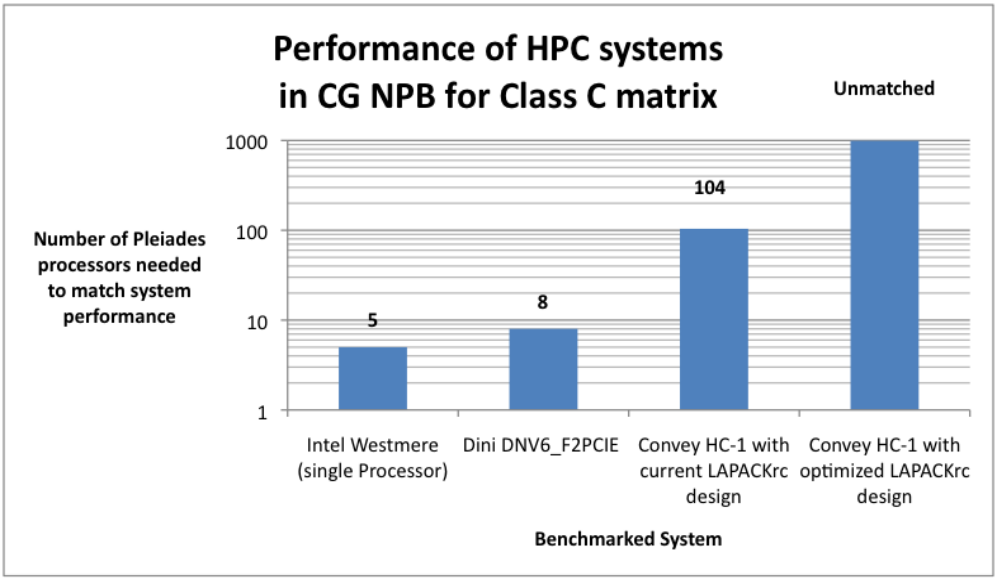

Figure 7. Number of processors from the Pleiades-wes supercomputer that are needed to match the performance of each of the commercial systems benchmarked in Figure 6.

Remarks on memory size and the ability of a SLAP to handle very large matrices

It is worth noting that the comparison above is relevant only when the size of the matrix at hand is such that it would fit inside the SLAP memory. If the memory available in the SLAP cannot accommodate the entire matrix, as it may be the case for example for the Class E CG NPB (currently under development by NASA), then the conclusion above may be inaccurate. However, if the reason to prefer a supercomputer over a SLAP is merely because of the supercomputer's ability to accommodate the problem in memory, and not because of its ability to solve the problem faster than the SLAP, there are alternative options that would be worth pursuing in order to enable the SLAP to solve the problem, namely:

(1) To build SLAP hardware that can accommodate very large increases in memory in a modular/incremental architecture and at low cost (given the low cost of memory today, this is an option that is both technically and economically feasible, and it is currently subject of Accelogic's R&D under the LAPACKrc™ program).

(2) To partition the problem into smaller sub-problems to be solved in sequence by the SLAP. For the CG case, this is achievable by partitioning the MVM operator into smaller MVM's produced by sub-matrices of the original matrix and sub-vectors of the relevant vectors.

[4] As per data provided by the NASA NPB team to Accelogic, February, 2011.

(3) To build more powerful SLAPs by composing networks of simpler SLAPs (e.g., a multi-FPGA/GPU system – this is also a subject of current R&D under the LAPACKrc™ program).

(4) To find an answer to the following open-research question:

Given the memory size limitations of the hardware at hand, how to partition the problem among the different processors in the supercomputer, including the SLAPs and any other available conventional processors, in such a way that the ultimate time-to-solution is minimal?

The answer to this question needs to consider the overall communication requirements of the algorithm partition, as well as the capacities of each processor and the complete topology and communication parameters of the supercomputer network, including the communication links to and from the SLAPs. This is, in our opinion, a fundamental question for the success of heterogeneous supercomputing and for the future of HPC in general. The question is currently the subject of intense R&D at Accelogic under the LAPACKrc™ program. Some of our preliminary results indicate that, for a wide class of algorithms, optimal partitions can be found through the solution of a pseudo-linear-programming problem based on a model of the network as a system. In many occasions, our results show that a single SLAP connected to the supercomputer through a high-bandwidth channel would be able to induce substantial reductions in time-to-solution, even if the SLAP solves only part of the computational problem being tackled by the supercomputer; although this requires some heavy data movement from and to the SLAP, the overall time-to-solution can be effectively reduced by outsourcing pieces of computation to the SLAP.[5]

5. CONCLUSIONS AND FUTURE WORK

Successful integration of Specialized Processors in HPC applications could potentially provide the processing performance necessary for tackling the most important computational challenges of this upcoming decade. Specialized Processors can produce "smart" specialized FLOPs at very high speeds and for a very low cost (both in terms of dollars and of power consumption), thus becoming a powerful complement to a supercomputer system.

We introduced the concept of Specialized Linear Algebra Processor (SLAP) as a low-cost Specialized Processor with the ability to compute certain specialized linear algebra kernels at speeds comparable to (or higher than) those achieved by general-purpose supercomputers. SLAPs can be placed strategically inside a supercomputer network to enhance its performance for certain applications, with relatively low investments in additional hardware or development costs. A SLAP could be a custom ASIC specialized in certain recurrent linear algebra computations, or it could be a commercial-off-the-shelf FPGA or GPU accelerator (or a small cluster of these), equipped with highly-optimized specialized linear algebra solvers that would be easily "callable" from the supercomputer network.

SLAPs are being designed by the authors to efficiently balance computing and communications for a wide family of linear algebra kernels/solvers. A demonstration of a NAS CG benchmark that makes use of the Krylov solver component of LAPACKrc™ in a state-of-the-art FPGA system indicates that, for certain NAS benchmarks, just a single SLAP could outperform an entire supercomputer.

[5] *J. Gonzalez, S. Fonseca, Network-adaptive resource allocator in a parallel computer system. U.S. Utility Patent Application, Accelogic, 2011.*

Given the ubiquity of sparse linear algebra in HPC, and the high volume of demand for computation of linear algebra kernels, we argue that it makes sense to invest in SLAP research programs that push on the efficiency of SLAPs further through specialization of both infrastructure and algorithms.

Future work includes the incorporation of additional features into the SLAP framework, including other types of solvers, as well as validation through the linear-algebra-intensive NPB application benchmarks (BT, SP, LU). Next generation SLAP implementations will address optimizations geared towards improved performance in the presence of latencies and communication bottlenecks that limit the scalability of large-scale codes when massive processing capacity is available. Communication avoidance algorithms, data compression, as well as optimized job schedulers specially tuned for exploiting SLAPs, are key for the success of Specialized Processors in HPC applications, and are also the subject of our most immediate research efforts.

Acknowledgments and Disclaimer

We wish to thank the NASA benchmarking personnel for their collaboration with NPB benchmark reports, as well as Jack Dongarra from the University of Tennessee, and John Casu from Chiral Dynamics Inc., for their consultation on this project. This material is based upon work supported by the U.S. Air Force and NASA, under award numbers FA9550-09-C-0052 and NNX10RA04C. Neither the U.S. Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the U.S. Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

REFERENCES

[1] A.R. Brodtkorb, C. Dyken, T.R. Hagen, J.M. Hjelmervik, O.O. Storaasli. 2010. State-of-the-art in heterogeneous computing. *Sci. Program.* 18, 1 (January 2010), pp.1-33

[2] F. Franchetti, M. Puschel, Y. Voronenko, S. Chellappa, J.M.F Moura, "Discrete fourier transform on multicore," *Signal Processing Magazine, IEEE* , vol.26, no.6, pp.90-102, November 2009.

[3] Y. Chen, X. Cui, H. Mei. Large-scale FFT on GPU clusters. In *Proc. of the 24th ACM International Conf. on Supercomputing (ICS '10)*. ACM, New York, NY, USA, 315-324. 2010.

[4] R. Strzodka, D. Goddeke. Pipelined Mixed Precision Algorithms on FPGAs for Fast and Accurate PDE Solvers from Low Precision Components. *IEEE Proceedings on Field-Programmable Custom Computing Machines*, 2006

[5] J. Gonzalez, S. Fonseca, R. Nunez. Achieving Maximal Concurrency in Heterogeneous HPC Systems –A case study on the efficient parallelization of the FFT algorithm in Multi-CPU/GPU systems. *SciDAC Conference 2011*. Denver, Colorado, July 2011.

[6] Ahern S, Alam S, Fahey M, Hartman-Baker R, Barrett R, Kendall R, Kothe D, Messer O, Mills R, Sankaran R, Tharrington A and White III J. 2007 Scientific Application Requirements for Leadership Computing at the Exascale. *Technical Report* (Oak Ridge National Laboratory)

[7] J. Gonzalez, R. C. Nunez. LAPACKrc™ : Fast linear algebra kernels/solvers for FPGA accelerators. *Scientific Discovery through Advanced Computing Program (SciDAC) Conference*, June 2009

[8] J. Gonzalez, R. C. Nunez. Extreme-speed scalable direct sparse solvers for heterogeneous CPU/GPU/FPGA supercomputing – an enhancement to the LAPACKrc™ library. *SciDAC Conference 2010*, Chattanooga, Tennessee, July 2010.

[9] Intel MKL Reference Manual, http://www.intel.com/software/products/mkl/docs/mklman.htm, 2006

[10] IMSL™ Numerical Libraries Family of Products. http://www.vni.com/products/imsl. Sept. 2006

[11] G. Golub, C. Van Loan, Matrix Computations, *The Johns Hopkins University Press*; 3 ed., 1996

[12] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou, Communication-avoiding parallel and

sequential QR and LU factorizations: theory and practice, *Tech. Report UCB/EECS-2008-89* UC Berkeley, EECS Department, 2008. LAWN #204.

[13] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L, Dagum, R Fatoohi, S. Fineberg, P Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan and S. Weeratunga. The NAS Parallel Benchmarks. *RNR Technical Report RNR-94-007*, March 1994

[14] DNV6_F2PCIe Godzilla's Part time Nanny ASIC Prototyping Engine Featuring Xilinx Virtex-6 Hosted via 4-lane PCI Express (GEN1). *Dini Group*. August, 2011. http://www.dinigroup.com/new/DNV6_F2PCIe.php.

[15] The Convey HC-1™: The World's First Hybrid-Core Computer. *Convey Compute*r. August, 2011. http://www.conveycomputer.com/Resources/HC-1%20Data%20Sheet.pdf

[16] David E. Shaw , Martin M. Deneroff , Ron O. Dror , Jeffrey S. Kuskin , Richard H. Larson , John K. Salmon , Cliff Young , Brannon Batson , Kevin J. Bowers , Jack C. Chao , Michael P. Eastwood , Joseph Gagliardo , J. P. Grossman , C. Richard Ho , Douglas J. Ierardi , István Kolossváry , John L. Klepeis , Timothy Layman , Christine McLeavey , Mark A. Moraes , Rolf Mueller , Edward C. Priest , Yibing Shan , Jochen Spengler , Michael Theobald , Brian Towles , Stanley C. Wang, Anton, a special-purpose machine for molecular dynamics simulation, *Communications of the ACM*, v.51 n.7, July 2008.

Appendix A. Specification of supercomputer systems for performance estimates.

(Data obtained from NASA's NAS Parallel Benchmark Team).

Machine Name	*Schirra*	*Columbia22*	*Rtjones*
Machine Type	IBM p575+	Altix4700	SGI ICE 8200
Processor	Power5+	Itanium2	Intel Clovertown
Model		9040 Montecito	Xeon X5355
Speed	1.9GHz	1.6GHz	2.66GHz
L2	1.92MB shared	256KB	8MB shared
L3	36MB shared	9MB	n/a
#Cores/Processor	8	2	4
#Cores/Node	16	2048	8
Memory/Core	2GB	1.9GB	1GB
Hyperthreads	2	n/a	n/a
TurboBoost	n/a	n/a	n/a
Compiler	IBM xlf 10.1	Intel 10.0.026	Intel 10.1.008
Comp-Flag	"-O3 -qhot"	"-O3"	"-O3 -xT -ipo"
MPI Library	poe-4.3	mpt-1.16.0.0	mpt-1.19
Processor Binding	launch.x	dplace -s1	dplace -s1

Machine Name	*Pleiades-har*	*Hyperwall2*	*Pleiades-neh*	*Pleiades-wes*
Machine Type	SGI ICE 8200EX	OpteronCluster	SGI ICE 8200EX	SGI ICE 8200EX
Processor	Intel Harpertown	AMD Opteron	Intel Nehalem-EP	Intel Westmere
Model	Xeon E5472	2354 Barcelona	Xeon X5570	Xeon X5670
Speed	3.0GHz	2.2GHz	2.93GHz	2.93GHz
L2	8MB shared	512KB	256KB	256KB
L3	n/a	2MB shared	8MB shared	12MB shared
#Cores/Processor	4	4	4	6
#Cores/Node	8	8	8	12
Memory/Core	1GB	2GB	3GB	2GB
Hyperthreads	n/a	n/a	2	2
TurboBoost	n/a	n/a	yes	yes
Compiler	Intel 10.1.015	GCC 4.1.2	Intel 11.1.046	Intel 11.1.046
Comp-Flag	"-O3 -xS -ipo"	"-O3 -mtune=amdfam10"	"-O3 -ip -xSSE4.2"	"-O3 -ip -xSSE4.2"
MPI Library	mpt-1.19	mvapich-0.9.9	mpt-1.25	mpt-1.25
Processor Binding	dplace -s1	mbind.x	dplace -s1	dplace -s1