Dynamic Load Balancing of Matrix-Vector Multiplications on Roadrunner Compute Nodes

José Carlos Sancho and Darren J. Kerbyson

Performance and Architecture Laboratory (PAL),
Los Alamos National Laboratory, NM 87545, USA
{jcsancho,djk}@lanl.gov

Abstract. Hybrid architectures that combine general purpose processors with accelerators are currently being adopted in several large-scale systems such as the petaflop Roadrunner supercomputer at Los Alamos. In this system, dual-core Opteron host processors are tightly coupled with PowerXCell 8i accelerator processors within each compute node. In this kind of hybrid architecture, an accelerated mode of operation is typically used to off-load performance hotspots in the computation to the accelerators. In this paper we explore the suitability of a variant of this acceleration mode in which the performance hotspots are actually shared between the host and the accelerators. To achieve this we have designed a new load balancing algorithm, which is optimized for the Roadrunner compute nodes, to dynamically distribute computation and associated data between the host and the accelerators at runtime. Results are presented using this approach, for sparse and dense matrix-vector multiplications, that show load-balancing can improve performance by up to 24% over solely using the accelerators.

1 Introduction

The unprecedent need for power efficiency has primarily driven the current design of hybrid computer architectures that combine traditional general purpose processors with specialized high-performance accelerators. Such a hybrid architecture has been recently installed at Los Alamos National Laboratory in the form of the Roadrunner supercomputer [1]. This system was the first to achieve a sustained performance of over 1 *PetaFlop/s* on the LINPACK benchmark.

In Roadrunner, dual-core Opteron host processors are tightly coupled with PowerXCell 8i processors [2]—an implementation of the Cell Broadband-Engine architecture (Cell BE) with high double-precision floating-point performance—within each compute node. This hybrid architecture can support several types of processing modes including: host-centric, accelerator-centric, and an accelerated mode of operation. The characteristics of an application determines which mode is most suitable. The host-centric mode can be thought of as the traditional mode of operation where applications solely use the host Opteron processors. In the accelerator-centric mode applications solely run on the PowerXCell 8i, an example that follows this mode can be found in the application VPIC [3],

H. Sips, D. Epema, and H.-X. Lin (Eds.): Euro-Par 2009, LNCS 5704, pp. 166–177, 2009.

Gordon Bell Prize finalist at SC08. In the accelerated mode, both the Opteron and PowerXCell 8i are used in such a way that performance-critical sections of computation are off-loaded to the PowerXCell 8i accelerators leaving the rest of the code to run on the host Opterons. SPaSM, a molecular dynamics code, is an example of an application that followed this accelerated approach [4], also Gordon Bell Prize finalist at SC08.

A variant of the accelerated mode is to share the performance hotspots between both the accelerator and host processors for simultaneous processing. The benefit of this is a potential gain in performance, since the computation power of both the host processors and accelerators can be harnessed simultaneously. The computation power of the host processors may be orders of magnitude smaller than that of the accelerators but at large-scale, including Roadrunner, the performance gain can be significant and thus should be exploited. However, this kind of accelerated mode increases complexity - extra tools are required in order to efficiently and dynamically load balance between the hosts and accelerators at runtime. Undertaking such a load balance during application execution is desirable in this context as it is difficult to determine costs associated with individual computations at compile-time, and also there may be changes in the amount of data to compute per processor during runtime which can result in repartitioning across nodes.

This paper addresses this challenge and presents a load-balancing algorithm in order to dynamically distribute the computation and associated data between the host Opterons and the PowerXCell 8i accelerators at runtime in the compute nodes of Roadrunner. For illustration purposes we address the common operation of matrix-vector multiplications on the form of $y_{i+1} = y_i + Ax$, where A is either a sparse or dense matrix and x and y are dense vectors. These operations are commonly found in scientific applications and are prime candidates to offload to accelerators. Results show that the dynamic load-balancing algorithm can improve the performance of these operations by up to 24% when using both host and accelerator processors in comparison to solely using the accelerators. In addition, the determination of the optimal load balance converges quickly taking only 7 iterations. Although the results as presented consider a 4-process parallel job (one compute node of Roadrunner), there is nothing to prevent our technique to being applied to larger process counts up to a system-wide parallel job because the scope of our technique is at the process level rather than at the system level.

The rest of this paper is organized as follows. Section 2 describes the architecture of a Roadrunner compute node. Section 3 describes our load-balancing algorithm. Section 4 includes experimental results from a Roadrunner node. Section 5 summarizes related work on matrix vector multiplications on hybrid architectures. And finally, conclusions from this work are given in Section 6.

2 The Roadrunner Compute Node

A compute node of Roadrunner is built using three compute blades (one IBM LS21 blade and two IBM QS22 blades) and one interconnect blade as shown in Figure 1. Each IBM LS21 blade contains two 1.8GHz dual-core Opteron

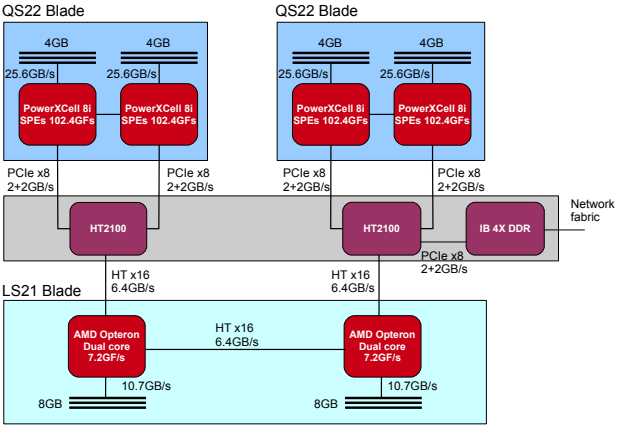

Fig. 1. The structure of a Roadrunner compute node

processors, and a single IBM QS22 blade contains two 3.2GHz PowerXCell 8i processors [2]. The fourth blade interconnects the three compute blades using two Broadcom HT2100 I/O controllers. These controllers convert the HyperTransport x16 connections from the Opterons to PCIe x8 buses — one to each PowerXCell 8i. In this configuration each Opteron core is uniquely paired with a different PowerXCell 8i processor when using the accelerated mode of operation.

The PowerXCell 8i processors have approximately 95% of the peak floating-point performance and 80% of the peak memory bandwidth of a node. Each PowerXCell 8i consists of eight Synergistic Processing Elements (SPEs) and one Power Processing Element (PPE). The eight SPEs have an aggregate peak performance of 102.4 GFlops/s (double-precision), or 204.8 Flops/s (single-precision) whereas dual-core Opteron has a peak performance of 7.2 GFlops/s (double-precision) or 14.4GFlops/s (single-precision). Therefore, a single PowerXCell 8i can potentially accelerate a compute-bound code by up to 28× (102.4/3.6) over a single Opteron core. In addition, each PowerXCell 8i processor has substantially more memory bandwidth than the Opterons, 25.6 GB/s compared to 10.7GB/s for a dual-core Opteron.

The PPE is a PowerPC processor core which runs a linux operating system (one per blade), and manages the SPEs. The SPEs are in-order execution processors with a two-way SIMD operation that do not have a cache. Instead they can directly access a 256KB high-speed memory called a *local store* which is explicitly accessed by direct memory access (DMA) transfers from the PPE memory space. Each compute node has a total of 32GB of memory, 8GB for each Opteron and 4GB for each PowerXCell 8i.

3 The Dynamic Load Balance Algorithm

In this section we describe our dynamic load-balancing algorithm applied to matrix-vector multiplication. These operations can be very time-consuming in

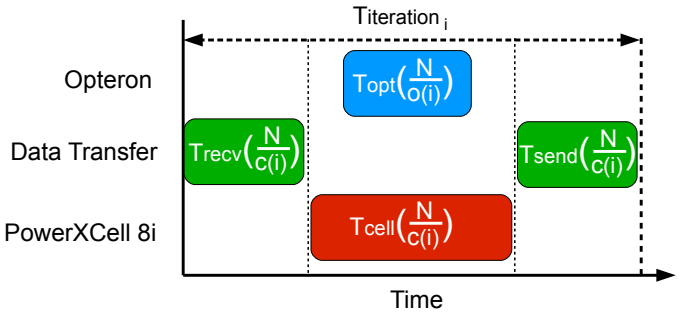

Fig. 2. Breakdown of iteration time (non-overlapping transfers)

codes such as iterative solvers where the matrix-vector multiplication, $y_{i+1} = y_i + Ax$, is performed once or more in each iteration of the application. We chose a single row of the matrix A as the smallest granularity of load balancing the data between the Opteron and PowerXCell 8i. The goal of the load-balancing algorithm is to find an optimal partitioning of the matrix rows to minimize the runtime when this calculation is performed multiple times as in iterative solvers. Formally, the load-balancing problem can be described as to minimize the following expression,

$$\sum_{i=1}^{n} max\left(T_{opt}(\frac{N}{o(i)}) + T_{trans}(\frac{N}{c(i)}), T_{cell}(\frac{N}{c(i)}) + T_{trans}(\frac{N}{c(i)})\right) + T_{house}(\frac{N}{c(i)})$$

when n is the number of times that the matrix-vector multiplication is performed; N is the number of matrix rows; $o(i)$ and $c(i)$ are the ratios in the amount of rows assigned to the Opteron and for the PowerXCell 8i, respectively, so $\frac{1}{o(i)} + \frac{1}{c(i)} = 1$; $T_{opt}(x)$ and $T_{cell}(x)$ are the times to perform the corresponding matrix-vector multiplications on x rows on the Opteron and PowerXCell 8i, respectively; $T_{trans}(x)$ is the sum of times for receiving data to compute on the PowerXCell 8i ($T_{recv}(x)$) and for sending back the results to the Opteron ($T_{send}(x)$); and finally, $T_{house}(x)$ is the *housekeeping* time associated with the execution of the load-balancing algorithm (described below) and formatting data for processing on the PowerXCell 8i. This formatting involves setting up of the various DMA transfers in order to iteratively transfer data in/out to/from SPEs and the replication of data structures in main memory in order to enforce the alignment of the DMA transfers. Therefore, the goal of the optimization problem is to dynamically define the function $o(i)$, $\forall i, 1 \leq i \leq n$ that minimizes the above expression; and hence, function $c(i)$ will be defined as $\frac{1}{1-\frac{1}{o(i)}}$.

We follow the operation of iterative solvers where the data that is transferred in and out of the operation in each iteration are the vectors y_i and y_{i+1} respectively. The matrix A is considered constant as in most iterative solvers, and hence it does not need to be transferred to the PowerXCell 8i each iteration. Similarly, the vector x also does not need to the transferred each iteration as it is computed internally based on the residuals.

The optimization problem can be simplified to the problem of minimizing $max\Big(T_{opt}(\frac{N}{o(i)}) + T_{trans}(\frac{N}{c(i)}), T_{cell}(\frac{N}{c(i)}) + T_{trans}(\frac{N}{c(i)})\Big)$ when the number of iterations is large enough that $T_{house}(\frac{N}{c(i)})$ is negligible. This case is illustrated in Figure 2 that shows the elapsed times on the Opteron, PowerXCell 8i, and the intranode-connection network in the particular case when data transfers are not overlapped with computation. Therefore, we want to minimize both $T_{opt}(\frac{N}{o(i)}) + T_{trans}(\frac{N}{c(i)})$ and $T_{cell}(\frac{N}{c(i)}) + T_{trans}(\frac{N}{c(i)})$ at the same time. By distributing the data carefully between processors (defining the function $o(x)$) it is possible to achieve the optimal balance that minimizes the above expression. For example, in the case that the PowerXCell 8i has too much to compute, we can move some data to the Opteron which reduces both $T_{cell}(\frac{N}{c(i)})$ and $T_{trans}(\frac{N}{c(i)})$ at expense of increasing $T_{opt}(\frac{N}{o(i)})$. Careful attention should be taken to prevent the case that the Opteron has too much data to compute, $T_{opt}(\frac{N}{o(i)}) > T_{cell}(\frac{N}{c(i)})$, which will also increase the iteration time. In the converse case, that the Opteron has too much data, some data can be moved from the Opteron to the PowerXCell 8i. Note again that assigning more data to the PowerXCell 8i in the next iteration, $i+1$, means that both the $T_{cell}(\frac{N}{c(i+1)})$ and $T_{trans}(\frac{N}{c(i+1)})$ will be increased. And therefore, the iteration time may be larger because the $T_{trans}(\frac{N}{c(i+1)})$ might be too high to offset the reduction in time on the Opteron, $T_{trans}(\frac{N}{c(i+1)}) - T_{trans}(\frac{N}{c(i)}) > T_{opt}(\frac{N}{o(i)}) - T_{opt}(\frac{N}{o(i+1)})$. It can also occur that the cell has to much data to compute with respect to the Opteron, $T_{cell}(\frac{N}{c(i+1)}) > T_{opt}(\frac{N}{o(i+1)})$.

On the other hand, when data transfers can be fully overlapped with computation, the load balancing is simplified to the case of making the compute-times on both the Opteron and PowerXCell 8i equal, $T_{opt}(\frac{N}{o(i)}) \simeq T_{cell}(\frac{N}{c(i)})$, in order to minimize the following expression $max(T_{opt}(\frac{N}{o(i)}), T_{cell}(\frac{N}{c(i)}))$. This is an ideal case that might be difficult to achieve in a real scenario because it depends on the application's data dependencies— data is not available yet because it needs to be combined with other data such as in the case of iterative solvers—, and the support of asynchronous operations on the communication system. Hence, the common scenario is that communications are only partially overlapped and the optimization problem described in Figure 2 applies.

The load-balancing algorithm proposed is based on combining the following three basic approaches: **accelerator-centric**, **performance-based**, and **trial-and-error** in order to converge at the optimal state as quickly as possible. This algorithm is comprised of five states as depicted in Figure 3. In the first state, we take an **accelerator-centric** approach where $o(1)$ is initialized to be $\frac{Peak_{cell}}{Peak_{opt}}$, where $Peak_{opt}$ and $Peak_{cell}$ are the peak flop performance of the PowerXCell 8i and Opteron, respectively. In Roadrunner, this is initialized to be 28, see Section 2. We use the peak performance of the processors as an starting point as this is available a priori. In principle, we do not know anything about the characteristics of the code and the peak flop performance is a safe alternative in

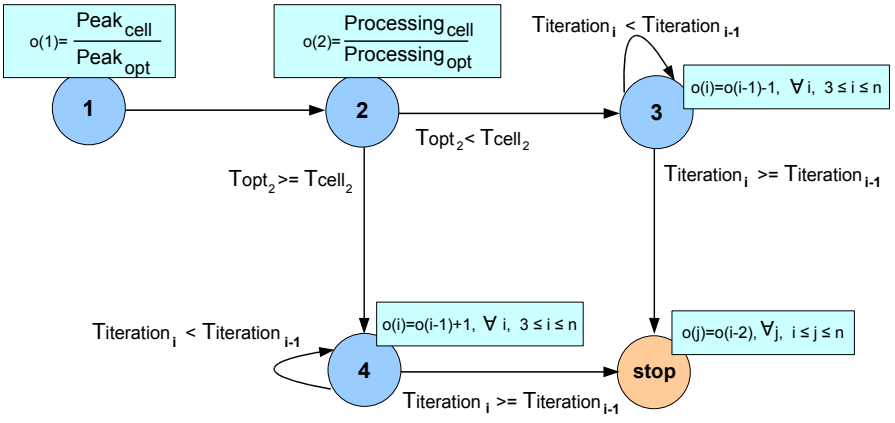

Fig. 3. States of the load-balance algorithm for n iterations

this architecture with respect to the peak memory bandwidth because most of the work will be performed on the PowerXCell 8i rather than the Opteron.

In the second state, we take a **performance-based** approach since we can collect actual timing information from the previous iteration. The principle of a performance-based approach is to distribute data based on how well the different processors perform, and thus allowing the algorithm to quickly converge to the optimal ratio. This is achieved by collecting the times, T_{opt} and T_{cell}, in order to calculate the processing rates, $Processing_{opt}$ and $Processing_{cell}$, for both the Opteron and PowerXCell 8i, thus $o(2) = \frac{Processing_{cell}}{Processing_{opt}}$. Note that T_{cell} and T_{trans} are measured independently instead of combining them into a single metric. This distinction is more efficient than the typical combination approach of as will be shown in the next section.

Finally, the third and fourth state are performed using a **trial-and-error** load-balancing strategy until the optimal balance is achieved. This is done by carefully assigning more or less data on the Opteron in order to not increase the $T_{iteration}$ for the next iteration. In particular, the third state is reached from the second state when the Opterons have not enough data to compute, case of $T_{opt_2} < T_{cell_2}$; and the forth state is reached from also the second state when the Opterons have too much data to compute, case of $T_{opt_2} \geq T_{cell_2}$. Note that these additional steps are not included in a typical performance-based load-balancing strategy, but they were necessary for the case of this particular architecture. In particular, the third state gradually decreases $o(i)$ by one assigning more data to the Opteron. Similarly, the fourth state gradually increases $o(i)$ by one assigning less data to the Opteron. Convergence is achieved when the current $T_{iteration_i}$ is higher than the previous $T_{iteration_{i-1}}$ time stopping the algorithm in state *stop*. In this state, we set up $o(j)$, $\forall j, i \leq j \leq n$ to the value used two iterations previously, $o(i-2)$. Note again, that for the case of fully overlapping transfers these additional states might not lead to the optimal balance as the second state should already give a good balance due to the fact that it is based on the

Table 1. Description of the matrices used in the evaluation

Name	Dimensions	Non-zeros	Description
Dense matrix	2K×2K	4M	Regular dense matrix
Sparse Harbor	47K×47K	2.37M	3D CFD of Charleston harbor
Sparse Dense	2K×2K	4M	Matrix in sparse format
Sparse Fluid	20.7K×20.7K	1.41M	Fluid structure interaction turbulence
Sparse QCD	49K×49K	1.90M	Quark propagators (QCD/LGT)
Sparse Ship	141K×141K	3.98M	FEM ship section/detail production
Sparse Cantiveler	62K×62K	4M	FEM cantiveler
Sparse Spheres	83K×83K	6M	FEM concentric spheres

achieved processing rate and the transfer time does not impact on the iteration time. However, these states are necessary in the case of partially overlapping and non-overlapping transfers where the transfer time does actually impact the iteration time.

4 Evaluation

We evaluate our load-balancing technique on a Roadrunner compute node as described in Section 2. A four-process parallel job— one process per each of the four Opteron cores in the Roadrunner node— was executed in the accelerated mode of operation. Each process performed the same double-precision floating-point matrix-vector multiplication several times. At the end of the calculation all the processes synchronize in order to account for the worst time. Timing data presented below are averages over multiple runs. We use the DaCS communication library [5] for communicating between Opteron and PowerXCell 8i processors, and OpenMPI version 1.3b [6] message passing library for the synchronization across Opteron cores. The Cell BE SDK version 3.1 was used to compile the code for the PowerXCell 8i processors.

We evaluated the performance of our load-balance technique, *Optimized balance*, as well as for the case of using our load-balance algorithm but considering T_{trans} in combination with T_{cell}, *Balance transfer*. Also for comparison purposes we evaluated the performance of using no load balancing in two cases: using only Opterons and using only PowerXCell 8i processors. In addition, we show results for a *Greedy* strategy that searches for the optimal load balance by exploring a wide range of distributions: it starts with the default distribution ($o(1) = \frac{Peak_{cell}}{Peak_{opt}}$) and gradually decrements it in steps of one every iteration to when all work is performed by the Opterons. The experiments were conducted on a dense matrix and on seven sparse matrices from a wide variety of actual applications as listed in Table 1 where Sparse Dense is a dense matrix, but formatted in the sparse format. We used the *Compressed Storage Row* (CSR) format [7] for defining the sparse matrices.

4.1 Results

Figure 4 shows the iteration time for the *Greedy*, *Optimized balance*, and the *Balance transfer* techniques on the sparse matrix Harbor. As can be seen, the minimum execution time is found at iteration 24 for the *Greedy* technique, where $o(24) = 5$. At this point the optimal load balance is achieved and the execution time is improved by 15% with respect to using the performance-based ratio ($o(2) = \frac{Processing_{cell}}{Processing_{opt}}$), and 3.4× with respect to $o(28) = 1$ where all the work is performed by the Opterons. The *Greedy* technique can easily find the optimal balance, but at the expense of a longer converge time (28 iterations) which is undesirable. In contrast, the *Optimized balance* technique converges faster and is able to find the optimal balance after only 5 iterations for this matrix. Converging faster is desirable as there is extra overhead due to housekeeping per iteration which could be significant, see Section 3. In the case of the Roadrunner compute node this time is around 60ms per iteration (results not shown). For the case of the *Balance transfer* technique we can see that the load-balance algorithm does not converge to the optimal solution. This is because including the T_{trans} in the T_{cell} makes the performance-based ratio too low ($o(2) = 2$) for this architecture due to T_{trans} being high. This forces the load-balancing search to stop too early as the next ratio tried in state 3 of the algorithm, unfortunately, does not use the accelerators at all ($o(3) = 1$), and hence $T_{iteration}$ is higher than the previous one.

Figure 5 illustrates how the *Optimized balance* technique converges to the optimal balance during the first 5 iterations by showing the corresponding times T_{opt}, T_{cell}, T_{trans}, and $T_{iteration}$ for each iteration. On the first iteration, T_{opt} is too small compared with the T_{cell} because $o(1) = \frac{Peak_{cell}}{Peak_{opt}}$, $o(1) = 28$ yields too little work for the Opterons compared with the PowerXCell 8i. On the second

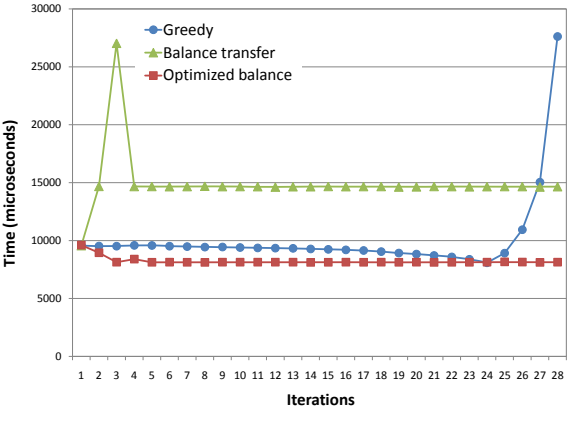

Fig. 4. Iteration time for the Greedy, Optimized balance, and Balance transfer techniques on the sparse matrix Harbor

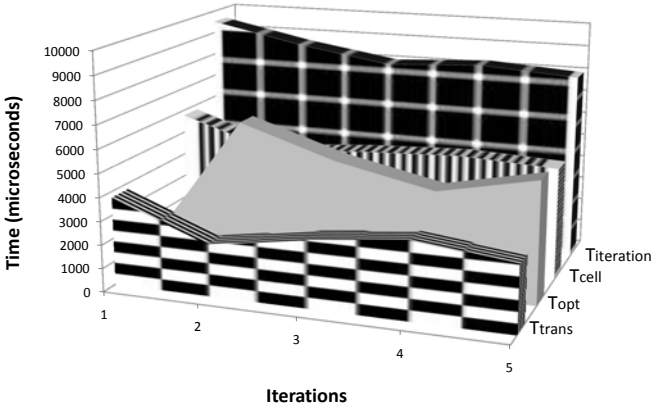

Fig. 5. Iteration time breakdown for the Optimized balance technique on the sparse matrix Harbor (first five iterations)

iteration, the ratio is already fixed to the current performance of the processors ($o(2) = \frac{Processing_{cell}}{Processing_{opt}}, o(2) = 4$), but actually results in too much work for the Opterons. On the third and fourth iterations, the load-balance algorithm is in state 4 gradually increasing the ratio ($o(3) = 5, o(4) = 6$) in order to reduce the work on the Opterons. During this, it is found that the third iteration results in a better $T_{iteration}$ time than the fourth iteration, and so the algorithm stops on the fifth iteration taking the best tested ratio, $o(3) = 5$, for subsequent iterations, $\forall i, 5 \leq i \leq n$. At the optimal balance, 65%, 60%, and 35% of the iteration time is spent on the T_{opt}, T_{cell}, and T_{trans}, respectively.

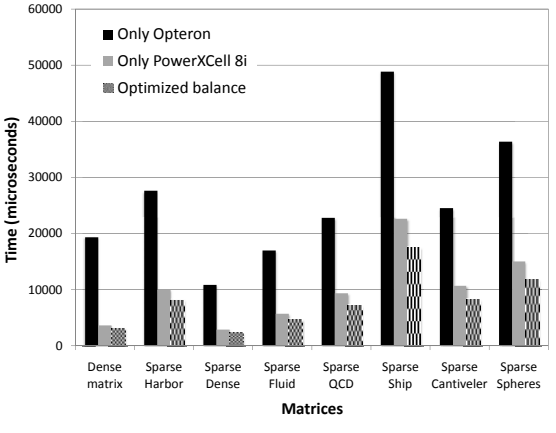

Fig. 6. Execution time for each matrix when using: only the Opteron, only the PowerXCell 8i, and the Optimized balancing technique

Figure 6 summarizes the execution iteration time for the suite of matrices evaluated when using *Optimized balance* (once the algorithm converged), when using the Opterons only and when using the PowerXCell 8is only. As can be seen, the *Optimized balance* achieves the best runtimes for all the matrices evaluated. In particular, for the dense matrix the performance improvement is 14% for the *Optimized balance* in comparison to using only the PowerXCell 8i. For the sparse matrices the improvements are 19%, 18%, 19%, 23%, 23%, 24%, 22% for the sparse matrices Harbor, Dense, Fluid, QCD, Ship, Cantiveler, and Spheres respectively. These improvements are mostly due to the fact that the computation of the sparse matrices is actually memory bound and thus take advantage of the relatively better memory performance of the Opterons rather than their flop performance. As expected, the improvements with respect to the Opteron are more noticeable, ranging from 4× on the sparse Fluid up to 6× for the dense matrix. Also, the number of iterations for the load-balancing algorithm to converge for these matrices is small— for the sparse matrices 5 iterations are required for convergence whereas the dense matrix required 7 iterations (results not shown).

5 Related Work

Matrix operations including sparse matrix-vector multiplications (SpMV) are key computational kernels in many scientific applications, and thus have been extensively studied. Today most work is focused on implementing these operations on emerging accelerator architectures including the Cell BE [8], FPGAs [9], and GPUs [10], as well as multi-core processors [8]. Although our SpMV implementation might not be so highly tuned for a particular processor in comparison to other implementations, they could be incorporated into our accelerator and host load-balancing method in order to improve overall performance.

On the other hand, there has been very little work on load-balancing matrix operations on hybrid (host-accelerator) architectures since typically they are fully offloaded to the accelerators. However, there is a significant work on load-balancing matrix operations like the SpMV on heterogeneous network of workstations (HNOWs) [11,12,13]. These systems are composed of non-uniform processors, network, and memory resources which partially resemble the hybrid platform studied in this work. For HNOWs most of the algorithms are optimized based on the characteristics of the target system. In fact as stated in [11] there is not a unique general solution for all platforms but rather different schemes are best for different applications and system configurations. This result is interesting because it suggests that there should be an efficient load balancing technique as well for our target platform. In particular, our platform is quite different from HNOWs. The processors are tightly attached to each other, so communications are much faster than in HNOWs. Also, there is a huge difference in the computing power of the processor types. These two features open new considerations in the design of load-balancing algorithms that they were not previously important. For example, in this new environment with fast communications it makes more sense to explore fine-grain load balancing algorithms, such as the one proposed in this paper, based on a trial-and-error strategies.

Additionally, in most of the load-balancing strategies for HNOWs distributing the load in proportion to the computing speed of the processors always leads to a perfectly balanced distribution [11,12]. However, we found that this strategy was not enough to achieve an optimal solution for the hybrid, host-accelerator architecture of Roadrunner.

Notwithstanding, it would be interesting to evaluate as future work the suitability of our proposed load-balancing algorithm to other hybrid platforms. In that regard, we could apply our dynamic load balancing technique into execution environments such as StarPU [14]—an unified execution model various accelerators— in order to dynamically determine the granularity of the tasks on different accelerators.

6 Conclusions

An optimized load-balancing algorithm has been presented in this paper to substantially increase the performance of a Roadrunner compute node. We have demonstrated that the proposed load-balance algorithm achieves a significant performance improvement, up to 24%, when simultaneously using both host (Opteron) and accelerator (PowerXCell 8i) processors in comparison to solely using the PowerXCell 8i processors in a traditional accelerated mode of operation. The load balancing was evaluated for matrix-vector multiplications which are commonly found in scientific applications.

These improvements come from the concurrent exploitation of the computation power of the host Opteron processors at the same time as the PowerXCell 8i accelerators for processing hotspot computations rather than uniquely offloading to the accelerators. These results suggest that the traditional accelerated mode of operation is not efficient enough to exploit the full potential of hybrid architectures including Roadrunner. With effective load-balancing techniques a more complex, but better accelerated mode of operation, can be enabled exploiting concurrently the full potential of all the available processors. In addition, the load-balance algorithm was carefully optimized to provide fast convergence time (7 iterations) making it sufficiently efficient to run during the execution of an application. This feature is desirable in order to dynamically adapt to the characteristics of the code, and thus it can potentially serve as a general load-balancing algorithm on this platform for other hotspot computations.

Acknowledgments

This work was funded in part by the Advanced Simulation and Computing program and the Office of Science of the Department of Energy. Los Alamos is operated by the Los Alamos National Security, LLC for the US Department of Energy under contract No. DE-AC52-06NA25396.

References

1. Barker, K.J., Davis, K., Hoisie, A., Kerbyson, D.J., Lang, M., Pakin, S., Sancho, J.C.: Entering the Petaflop Era: The Architecture and Performance of Roadrunner. In: SC 2008, Austin, TX (2008)
2. Kahle, J.A., Day, M.N., Hofstee, H.P., Johns, C.R., Maeurer, T.R., Shippy, D.: Introduction to the cell multiprocessor. IBM Journal of Research and Development 49(4), 589–604 (2005)
3. Bowers, K.J., Albright, B.J., Bergen, B.K., Yin, L., Barker, K.J., Kerbyson, D.J.: 0.374 Pflop/s Trillion-particle Particle-in-cell Modeling of Laser Plasma Interactions on Roadrunner. In: ACM Gordon Bell Prize finalist, Supercomputing Conference (SC 2008), Austin, TX (2008)
4. Swaminarayan, S., Kadau, K., Germanm, T.C.: 350-450 tflops molecular dynamics simulations on the roadrunner general-purpose heterogeneous supercomputer. In: ACM Gordon Bell Prize finalist, Supercomputing Conference (SC 2008), Austin, TX (2008)
5. IBM: Data Communication and Synchronization Library for Hybrid-x86: Programmer's Guide and API Reference. IBM Technical document SC33-8408-00, IBM SDK for Multicore Acceleration version 3, release 0 (2007)
6. Indiana University: Open-MPI (2009), `http://www.open-mpi.org`
7. Barrett, R., Berry, M., Chan, T.F., Demmel, J., Donato, J.M., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., der Vorst, H.V.: Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd edn. SIAM, Philadelphia (1994)
8. Williams, S., Oliker, L., Vuduc, R., Demmel, J., Yelick, K.: Optimization of sparse matrix-vector multiplication on emerging multicore platforms. In: Supercomputing Conference (SC 2007), Reno, NV (2007)
9. Morris, G.R., Prasanna, V.K.: Sparse matrix computations on reconfigurable hardware. IEEE Computer 40(3), 58–64 (2007)
10. Garland, M.: Sparse matrix computations on manycore GPU's. In: Annual ACM IEEE Design Automation Conference (DAC 2008), Anaheim, CA (2008)
11. Zaki, M.J., Li, W., Parthasarathy, S.: Customized dynamic load balancing for a network of workstations. Parallel and Distributed Computing 43(2), 156–162 (1997)
12. Pineau, J.F., Robert, Y., Vivie, F.: Revisiting matrix product on master-worker platforms. In: Workshop on Advances in Parallel and Distributed Computational Models (APDCM 2007), IEEE International Parallel and Distributed Processing Symposium, Long Beach, CA (2007)
13. Xu, C., Lau, F.: Load Balancing in Parallel Computers: Theory and Practice. Kluwer Academic Publishers, Norwell (1996)
14. Augonnet, C., Thibault, S., Namyst, R., Wacrenier, P.: StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. In: Euro-Par Conference, Delft, The Netherlands (2009)