Accelerating Agent-Based Ecosystem Models using the Cell Broadband Engine

Michael Lange, Tony Field

▶ To cite this version:

Michael Lange, Tony Field. Accelerating Agent-Based Ecosystem Models using the Cell Broadband Engine. A4MMC 2010 - 1st Workshop on Applications for Multi and Many Core Processors, Jun 2010, Saint Malo, France. inria-00493886

HAL Id: inria-00493886
https://inria.hal.science/inria-00493886

Submitted on 21 Jun 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Accelerating Agent-Based Ecosystem Models using the Cell Broadband Engine

Michael Lange and Tony Field
{michael.lange,t.field}@imperial.ac.uk

Department of Computing
Imperial College London

Abstract. This paper investigates how the parallel streaming capabilities of the Cell Broadband Engine can be used to speed up a class of agent-based plankton models generated from a domain-specific model compiler called the Virtual Ecology Workbench (VEW). We show that excellent speed-ups over a conventional x86 platform can be achieved for the agent update loop. We also show that scalability of the application as a whole is limited by the need to perform particle management, which splits and merges agents in order to keep the global agent count within specified bounds. Furthermore, we identify the size of the PPE L2 cache as the main hardware limitation for this process and give an indication of how to perform the required searches more efficiently.

1 Introduction

Multi-core stream processors have been used to provide significant performance gains for scientific algorithms in recent years [3]. The Cell Broadband Engine in particular has shown great promise for accelerating scientific applications due to its Stream Processing architecture [10] [11]. This paper investigates the use of the Cell Broadband Engine to accelerate a class of *agent-based* Virtual Ecology models for use in oceanographic research.

We focus on models generated by the Virtual Ecology Workbench, a domain-specific model compiler used by oceanographers to build and analyze models of plankton ecosystems in the upper ocean. VEW models have been used to study complex cause-effect relationships in ocean ecosystems, contributing to our understanding of the biodiversity of the ocean. The effect of marine plankton on the atmosphere and global climate, as well as the plankton ecosystem's response to external influences, like fishing or pollution, have also been the focus of VEW-assisted studies [9] [5].

Models generated by the VEW simulate large numbers of plankton particles following the Lagrangian Ensemble (LE) metamodel [12], an *individual-based* approach that uses agents to simulate the biological and bio-chemical behaviour of individual plankters. In contrast to traditional *population-based* models this allows for a detailed analysis of life-histories of individual micro-organisms. LE modelling further facilitates the study of emergent properties of the ecosystem as a whole and avoids chaotic instabilities often observed in *population-based* models [13].

The main workload of VEW-generated models involves updating large numbers of agents during each simulation timestep. Since the updates are independent of each other the process can, in principle, benefit greatly from parallel processing. Attempts to accelerate VEW models using FPGAs [7] have shown significant speedups, prompting further investigation of dedicated hardware solutions for parallel VEW simulations.

This paper investigates the use of Stream Processing principles to accelerate VEW models on the Cell processor, as well as the performance limitations imposed by agent management functions used in the VEW algorithm. We focus on efficient parallel agent update computation and make the following contributions:

- We describe a framework for parallel agent update on the Cell's SPE accelerator cores in Section 4. Our design includes buffered processing of agent data streams, as well as meta-data exchange to provide centralized scheduling. In addition to an implemented prototype simulation we also outline an extension to the VEW model compiler that generates SPU-specific SIMD vector code for performing agent update arithmetic.
- An evaluation of the achieved performance gains based on a simple characteristic VEW simulation is given in Section 5. We show that the parallel speedups observed for the agent update loop, although promising, are limited by sequential overheads due to agent management functions.
- We present an analysis of the observed limitations in Section 6. This shows that bottlenecks are caused by inefficient cache utilization. In this section we also present a hybrid processing approach that overcomes the identified limitations by exchanging meta-data between SPE and PPE processors to handle data searches and copy operations more efficiently. We demonstrate this method on a simple sequential search routine imposed through one of our design choices and largely eliminate the sequential overhead.
- We discuss the implications of our findings for future research in Section 7. In particular we focus on the current *particle management* function which is shown to limit parallel scalability on platforms such as the Cell.

2 Background

The Virtual Ecology Workbench (VEW) is a software tool designed to aid the creation and analysis of virtual plankton ecosystems based on the Lagrangian Ensemble (LE) metamodel [12]. These models are *individual-based* and use agents (sometimes referred to as 'particles') to simulate large numbers of plankters and the various feedback processes needed to study oceanic ecosystems and their various emergent demographic properties.

Since it is impossible to model every organism in the ocean individually, each agent in LE models represents a *sub-population* of identical plankton particles modelled after the behaviour of one individual plankter. Each agent follows its own trajectory and keeps an associated sub-population size in addition to internal state variables that are application specific. The sub-population size is used to infer demographic properties of the ecosystem, such as the global population count and the concentration of a particular species in a given domain.

The VEW provides oceanographers with an easy way of specifying classes of plankton species in terms of primitive phenotypic equations which are scientifically sound in that they are based on reproducible laboratory experiments. These equations define the basic behaviour for individual plankters in response to local environmental properties [6], for example light, temperature, nutrient concentration etc. An agent's internal state is then updated using equations relevant to its current biological state.

After specifying plankton species and additional environment parameterizations, the VEW automatically creates code from a high-level domain-specific mathematical modelling language called *Planktonica* [5] and provides further tools for the analysis of output data. The VEW framework substantially raises the level of abstraction in model development and facilitates the creation and analysis of complex virtual ecosystems without the need for conventional programming. It is an example of a domain-specific problem solving environment.

VEW models simulate virtual plankton ecosystems in a one-dimensional virtual water column that extends from the ocean surface to a specified depth (typically $500m$). The column is divided into layers of $1m$ depth and currently ignores horizontal water fluxes; one imagines the column to be sealed on all sides with an open boundary at the bottom. The upper metre of the column is sub-divided into more fine-grained layers in order to accurately model light absorption near the sea surface.

The combination of solar heating and cooling to the atmosphere, combined with atmospheric wind stresses and other environmental processes determine the depth of the turbulent *Mixing Layer* in the upper ocean – the so-called *Turbocline*. The depth of the turbocline varies both diurnally and seasonally. Particles above the turbocline are subject to turbulent mixing, which is approximated in the model by random displacement[1]. Below the turbocline there is laminar flow; here agents sink under the influence of gravity, although they may subsequently be 're-entrained' into the mixing layer, should the turbocline later deepen sufficiently to catch up with them. The dynamics of individual plankters is heavily influenced by turbulence.

Phytoplankton obtain energy from the sun through photosynthesis and absorb light in the process. Collectively, the plankton influence the profile of light and temperature in the column and this, in turn, affects the physics of turbulence – a process known as *biofeedback*.

VEW-created models simulate inter-agent processes, such as predation and ingestion, although these are not modelled using interactions between individual agents (an $O(n^2)$ process). Instead, the population of a particular plankton type is aggregated into a *field* (a concentration for each layer); individual agents then

[1] This approximation is good provided the time step is sufficiently large; we use a 30-minute time step

interact with these fields (an $O(n)$ process). This is the essence of the Lagrangian Ensemble approach.

Although agent updates can, in principle, proceed independently, the various fields (including nutrient fields) are shared. Any changes to these fields resulting from predation and the uptake or release of nutrients thus has to be negotiated among the agents. Furthermore, predation may reduce the concentration of a particular species which must, in turn, be reflected by a change in the agent's internal population counts. These processes are carefully managed in the generated code so that corrections from the previous time step and updates relevant to the current time step are merged into a single piece of 'update' code for each agent type.

The demographic noise, i.e. the statistical variability in the populations observed among identical runs of the model but with different random numbers, is limited by ensuring that a sufficient number of agents exists in the water column. For this purpose a *Particle Management* (PM) function is executed between timesteps. This splits agents with the largest sub-populations, creating new agents with independent trajectory. Similarly, in order to limit the computational cost of an over-populated simulation, the PM may also merge the least populated agents. The PM process thus provides a trade-off between computational cost and statistical accuracy for agent-based models.

3 Hardware

The Cell Broadband Engine is a heterogeneous stream processor which features a unique cache hierarchy. It consists of a central general- purpose CPU, the Power Processing Element (PPE), and eight vector processing nodes called Synergistic Processing Elements (SPE). These are much simpler in their architecture, but are optimized for high-throughput floating point arithmetic.

The PPE is a dual-threaded single-core general-purpose processor based on IBM's PowerPC architecture. It uses a typical CPU cache hierarchy including a 512KB L2 write-back cache with 128-bit cache lines. In our simulation the PPE will delegate the main workload of updating agents to the SPE arithmetic cores, whilst maintaining attributes of the water column in memory. In addition it is also responsible for the *particle management* task of the algorithm.

A circular on-chip data bus connects the PPE with the SPEs and transports data streams from main memory to the SPE accelerator nodes. The bus provides the high data bandwidth required for Stream Processing and has been shown to transport up to 196GB/sec in experiments with near perfect utilization [2]. In practice, however, memory bandwidth is limited by the 25.6GB/sec XDR main memory interface.

The SPE accelerator nodes provide the computational power of the Cell through vectorized SIMD computation at 3.2GHz core frequency. Each SPE comprises a 256KB Local Store (LS) memory area for buffering stream data, as well as executable code. A separate Memory Flow Controller (MFC) coordinates asynchronous data exchange with main memory for efficient data localization.

The Synergistic Processing Unit (SPU) is the functional core of the SPE. It is optimized for SIMD vector arithmetic on vectors of four *single-precision floats* and uses instruction pre-fetching on two separate pipelines for Load/Store and arithmetic instructions. Although it is possible to use double-precision floats, these are not implemented natively and are therefore more costly. The original VEW works with double-precision, but we decided to only use single-precision in our prototype in order to investigate the performance limitations of the Cell processor available in the Playstation3.

The SPU uses branch-prediction in order to optimize loop constructs. The compiler predicts continued execution of the loop body, resulting in a 19 cycle penalty to exit the loop due to a pipeline flush. In order to avoid unnecessary pipeline flushes for conditional vector statements, outcomes for both branches are typically pre-computed. The results are then interleaved according to a vector of flags indicating the outcome of the condition for each element.

This particular style of execution demands a special-purpose instruction set, provided by IBM's Cell SDK through intrinsic functions. These include memory flow controls for asynchronous Direct Memory Access (DMA) via the MFC, message-passing primitives for communication between PPE and SPE (mailboxes), and vector instructions for arithmetic calculations and branch prediction.

4 Implementation

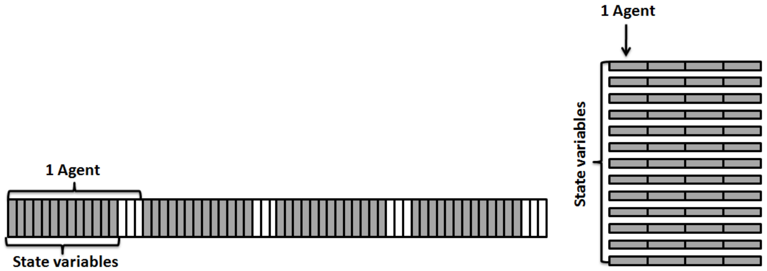
The main body of a VEW simulation comprises an agent update loop, which is the main source of parallelism in the model, and various housekeeping tasks that are predominantly sequential. Processing agent updates in parallel requires an efficient means of localizing agent data to the local SPE memory, as well as the exploitation of SIMD vector code for performing agent updates.

Memory Layout

Two important attributes of each agent are its associated *species* and its growth *stage*; these collectively define its *type*. There is a different update kernel for each type. To prepare agents for parallel update on the Cell they are stored as *type-homogeneous* agent arrays in memory.

Agent update is complicated by the fact that an agent may change its type at the end of a time step. For example, it may evolve to the next growth stage or it might die. Furthermore, agents may spawn new agents, for example as a result of reproduction. These various processes mean that agents may have to be moved to a different array before the start of the next timestep. This requires an additional search through each agent array during the sequential part of the algorithm. However, we will use this additional search routine to demonstrate how to combine the capabilities of PPE and SPEs to solve agent management tasks more elegantly (Section 6).

Due to the Cell's 128-bit bus lanes agents need to be aligned to multiples of 4 floating point numbers. In the prototype simulation an agent's state consists of 13 variables, requiring 16 `floats` to be allocated and stored in main memory. Conceptually this corresponds to an Array of Structures (AOS) layout (Figure 1(a)), which is maintained during the DMA transfer to the SPEs.



(a) AOS layout used in memory and for DMA transfers

(b) SOA layout used during vector processing

Fig. 1. Agent data layout in global and local memory.

Vector processing on the SPU, on the other hand, requires a Structure of Arrays (SOA) layout, where each vector holds the same variable from four respective agents (Figure 1(b)). After buffering agents in the LS we therefore need to convert the agent state variables, as well as the environment variables, to `vector float` types in a small loop.

Since we are mostly processing four agents at a time, we can unroll this conversion loop, which allows the compiler to optimize the process through inlining. For conversion of less than four agents the loop accesses each element in all vectors in turn, allowing us to ignore the unused elements in incomplete vectors. This scheme is similar to a *pad-with-zeros* approach and is also used when preparing the data for export to main memory.

Agent Allocation

Since our framework aims to utilize the SPEs as pure Stream Processing nodes, the PPE handles all agent allocation and the scheduling of update tasks to the SPEs. This also provides the possibility of investigating different load-balancing schemes. A *descriptor* data structure is used to transport meta-data to the SPEs, which contains the memory address and size of each agent block. A two-dimensional array of *descriptor* meta-objects is held in memory, mirroring the actual maintained agent blocks one-to-one. In addition we can use similar *feedback* objects to send meta-data back to main memory after processing a block of agents.

This meta-data exchange allows us to control SPE processing in a non-static fashion and write an independent scheduling function that allocates a *descriptor* block to each SPE. We found an Round-Robin allocation scheme for independent agent arrays to provide sufficient load-balance between SPE processing nodes.

Agent Localization

The implemented data transfer framework follows general Stream Processing principles by masking all data transfer latencies through asynchronous DMA. By using three agent buffers we allow GET and PUT transfers to happen in parallel to SPU computation, allowing for continuous SPU execution.

This *Triple-Buffering* approach proved sufficient to handle all data transfers without incurring unexpected data stalls. The actual size of agent blocks showed no noticeable effect on DMA transfer times, as long as a minimum of 2 agents per block is maintained.

Since Stream Processing requires the agent data to be pre-fetched, we need to load the corresponding *descriptor* block before initializing the data GET. Thus we use a *2-step-lookahead* loop to first fetch the meta-information, followed by the agent data, whilst maintaining the triple-buffered loop structure. This is illustrated in Figure 2. The different data and meta buffers are synchronized over a common counter (n).

```
Load shared environment data
Read 2 initial descriptors
GET first agent block
while(work to do){
  schedule agent GET to buffer n + 1
  schedule descriptor fetch for block n + 2
  process agent block in buffer n
  schedule agent PUT from buffer n
  schedule feedback store for block n
}
Store environment data
Process and PUT last agent block
```

Fig. 2. Pseudo code for SPE execution loop.

Agent Updates

We extended the existing VEW code generator module to produce optimized SPU agent update code that can be compiled and executed with the designed agent transfer framework. This allowed us to generate inlined vector intrinsics resulting in fast agent update kernels with low-level optimizations.

Particular care needs to be taken when generating conditional statements, where we used pre-calculation of both branches, as described in Section 3. Through recursive compilation the model compiler is capable of handling nested conditionals of any depth.

5 Performance Evaluation

Agents	Parallel runtime	Sequential runtime		Parallel speedup	
	Cell	1.6GHz	3.2GHz	1.6GHz	3.2GHz
4000	5.4s	96.7s	30.5s	17.91	5.65
8000	9.5s	187.6s	58s	19.75	6.11
16000	17.5s	368.3s	114.3s	21.05	6.53
32000	34s	752.2s	230.8s	22.12	6.79
64000	66.7s	1478.7s	454.1s	22.17	6.81

Table 1. Runtimes of agent updates on Cell and x86 with respective parallel speedup.

The non-standard architecture of the Cell makes it hard to compare accurately its performance to conventional x86 CPU architectures. For our performance evaluation we therefore used a 1.6GHz dual-core, as well as a 3.2GHz quad-core x86 CPU to run a single-threaded sequential version of our prototype simulation.

The Cell implementation was developed and tested on a Playstation3 with 6 enabled SPEs, running Ubuntu 8.10 with IBM's Cell SDK and compilers [2]. A single-precision floating point version of the prototype model was used for both architectures [3].

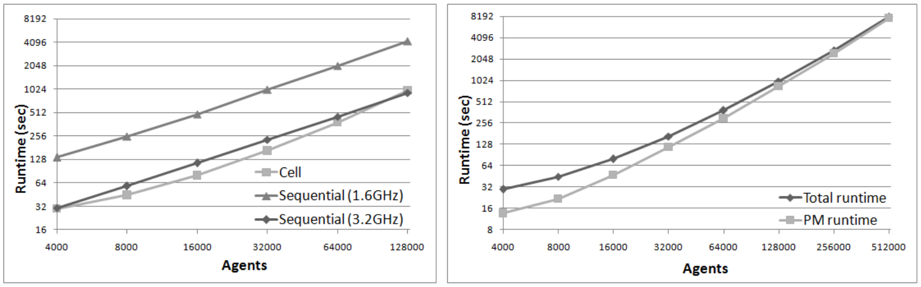
For the purpose of this investigation we used a simplified version of the LERM [9] model, which simulates a Diatom (a type of marine phytoplankton) population in the Azores region for 2 years. This so-called *Toymodel* simulation incorporates all major components of the VEW algorithm and was handcoded for the Cell. For comparison we used an identical single-threaded sequential version written in C on the x86 platforms.

We scaled the size of the simulation by increasing the initial number of agents. Considering just the agent updates (i.e. turning off particle management), we observed average parallel speedups of more than 22 over the 1.6GHz runs of the simulation and more than 6 over the 3.2GHz version (Table 1). Furthermore, we verified that agent update runtimes scale approximately linearly with increasing numbers of SPE cores.

[2] A PowerXCell i8, which provides improved double-precision floating point performance, was not available to us during this project.

[3] The use of single-precision only adds a small amount of random noise to an already noisy simulation.

Figure 3(a) shows the total execution times, i.e. including particle management and various other sequential components. The graphs show the Cell is significantly faster than the 1.6GHz CPU but fails to provide significant overall speedup over the 3.2GHz processor.



(a) Total execution times on Cell and x86 CPUs

(b) Total execution time and individual *particle manager* runtime on Cell

Fig. 3. Execution times of parallel and sequential prototype.

The overall non-linear scalability of the Cell version of the code is also quite evident. This is due to the particle manager, which is the most significant sequential component running on the PPE, as shown in Figure 3(b).

6 More on Particle Management

Particle management consists of two main search functions, *Split* and *Merge*, that exhaustively search the agent arrays in order to identify the agents that need to be altered. The *Merge* function furthermore includes a $O(n^2)$ lookup in order to find the relevant pairs for merging. The overall runtime shows a non-linear trend, although there is a linear number invocations of the *Split* and *Merge* functions.

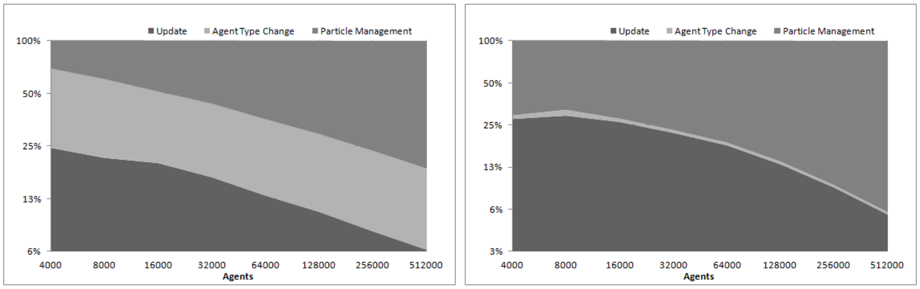
Due to the AOS layout in main memory copying agents can be handled very efficiently on the PPE, since the L2 cache will already contain the whole agent once the first field is accessed. Similarly, the copied agent will be written to memory in one block due to the write-back property of the cache. On the other hand, searching through all agents whilst inspecting only one variable will be very inefficient since each agent is still loaded anyway. However, once the simulation contains more agents than the L2 cache can hold, each agent will be read multiple times during the *Merge* process($O(n^2)$), resulting in poor cache performance.

The size of the PPE's L2 cache therefore poses the main hardware limitation. The cache can hold 512KB which corresponds to 8K agents of 16 `floats` (64B), in contrast to the 2MB cache found on the x86 CPU, which may hold up to 32K agents.

In the x86 version of the code each timestep is dominated by the agent update loop, so the overhead of the particle manager is less evident. In the Cell version the cost of the update loop is substantially reduced and the cost of sequential computations on the PPE is increased when compared with an x86, as described above. The two combined have the effect of increasing the dominance of particle manager (see also Figure 4(b)), as per Amdahl's law.

Agent Type Changes

The same cache limitations can be observed when handling agent type changes on the PPE. As described in Section 4, an agent may arbitrarily change its growth stage during the update process. Thus we need to move all agents that did change type to the correct sub-array to ensure *type-homogeneity* before the next timestep. In contrast to particle management, however, we only need to traverse each sub-array once to identify all agents to be moved. We can therefore flag all state changes on the SPEs as they happen, and use the *feedback* metadata objects to send back the indices of the affected agents. Knowing the exact location of all agents that need to be moved, we can then utilize the PPE cache more efficiently to copy the agents, avoiding irregular memory access patterns.



(a) Linear search agent copy

(b) Indexed agent copy

Fig. 4. Percentage of runtime spent in sequential and parallel parts of the main loop.

The effect of this method can be seen from Figure 4, which shows the respective contribution of parallel and sequential components. Figure 4(a) highlights the significant amount of time spent performing the search on the PPE, while Figure 4(b) shows improvements resulting from the parallel search strategy on the SPEs. This demonstrates that the SPE-assisted agent copy method substantially reduces the overhead imposed by this component, indicating how we can utilize the advantages of both types of processing core to perform agent management tasks very efficiently.

Unfortunately, the same ideas can not be used to optimize the particle management code. This is because the current management rules limit the total agent count globally. During parallel processing each SPE node only sees a local subset of all agents, and is therefore not able to tell how many agents it has to flag for *Split/Merge* copy operations.

7 Future Work

The original intention of this work was to explore ways to parallelize VEW models with hope of providing linear scalability with increasing numbers of agents. This was done in in order to prepare the VEW algorithm to be applied to three dimensional scenarios, which will inevitably feature agent counts several orders of magnitude larger than the existing models.

The greatest obstacle to parallelizing the VEW algorithm is the particle manager. The current rules are applied globally and thus do not scale well. We are currently exploring alternative management strategies that predominantly involve local computations and which avoid the need for sophisticated search strategies, e.g. to find the k largest or smallest agents by population count. The evaluation of new particle management algorithms involves an extensive combination of performance analysis and statistical analysis that is by no means trivial.

8 Conclusion

We have shown how to utilize Stream Processing capabilities of the Cell Broadband Engine to parallelize agent-based ecosystem models based upon the Lagrangian Ensemble metamodel. We have demonstrated parallel speedups for agent update computation of over 22 and 6 when compared to a 1.6GHz and 3.2GHz reference CPU respectively. We have also shown that the Cell can handle models with significantly large numbers of agents. The performance gains achieved are similar to speedups achieved for other scientific codes [8] [4] and indicate that the Cell processor has considerable potential for accelerating this class of applications.

We have found that the Cell DMA system can be used effectively to perform meta-data exchange in addition to agent data streaming. This adds flexibility to the overall design and allows us to perform SPE-based searches in conjunction with agent update computation. This approach overcomes one of the major drawbacks of SPE computation, which has been shown to work inefficiently for SPE-based searches without sufficient additional workload [1].

Through our investigation we also highlight that the particle management process, which is currently implemented sequentially, becomes a significant bottleneck. This highlights the importance of parallelizing the particle management algorithm.

Our investigation also demonstrates that the size of the PPE L2 cache can significantly limit performance. Since scientific applications often require several different memory access patterns the heterogeneity of the Cell's individual cores can be a great advantage. For example, randomly copying coherent data structures such as LE agents profits from the conventional cache hierarchy of the PPE, while exhaustive searches can be performed efficiently on the SPEs if combined with sufficient arithmetic workload. However, as shown in the previous section, the current implementation of the PPE does not compare well to conventional general-purpose CPUs, limiting its use as an additional processing core.

References

[1] D. A. Bader, V. Agarwal, and K. Madduri. On the Design and Analysis of Irregular Algorithms on the Cell Processor: A Case Study of List Ranking. *Proceedings of 21st IEEE IPDPS*, 2007.

[2] T. Chen, R. Raghavan, J. Dale, and E. Iwata. *Cell Broadband Engine Architecture and its first implementation.* URL http://www.ibm.com/developerworks/power/library/pa-cellperf/.

[3] M. Erez, J. H. Ahn, J. Gummaraju, M. Rosenblum, and W. J. Dally. Executing irregular scientific applications on stream architectures. *ACM*, 2007.

[4] G. De Fabritiis. Performance of the Cell processor for biomolecular simulations. *Computer Physics Communications*, 176:660–664, 2007.

[5] W. R. Hinsley. *Planktonica: A system for doing biological oceanography by computer.* PhD thesis, Imperial College London, 2005.

[6] W. R. Hinsley, A. J. Field, and J. D. Woods. Creating Individual-based Models of the Plankton Ecosystem. *Lectures in Computer Science*, 4487: 111–118, 2007.

[7] J. Lamoureux, T. Field, and W. Luk. Accelerating a Virtual Ecology Model with FPGAs. *20th International Conference on Application-specific Systems, Architectures and Processors*, 2009.

[8] F. Petrini, G. Fossum, J. Fernandez, A. L. Varbanescu, M. Kistler, and M. Perrone. Multicore surprises: Lessons learned from optimizing sweep3d on the cell broadband engine. *Proceedings of 21st IEEE IPDPS*, 2007.

[9] M. Sinerchia. *Testing theories on fisheries recruitment.* PhD thesis, Imperial College London, 2007.

[10] J. Spray, J. Hill, and A. Trew. Performance of a Lattice Quantum Chronodynamics kernel on the Cell processor. *Computer Physics Communications*, 179:642–646, 2008.

[11] S. Williams, J. Shalf, L. Oliker, S. Kamil, P. Husbands, and K. Yelick. The Potential of the Cell Processor for Scientific Computing. *ACM*, 2005.

[12] J. D. Woods. The Lagrangian Ensemble metamodel for simulating plankton ecosystems. *Progress in Oceanography*, 67:84–159, 2005.

[13] J. D. Woods, A. Perilli, and W. Barkmann. Stability and predictability of a virtual plankton ecosystem created with an individual-based model. *Progress in Oceanography*, 67:43–83, 2005.