Cluster Comput (2011) 14:419–432
DOI 10.1007/s10586-011-0169-4

Particle simulation on the Cell BE architecture

**Betul Demiroz · Haluk R. Topcuoglu ·
Mahmut Kandemir · Oguz Tosun**

Received: 9 November 2010 / Accepted: 30 June 2011 / Published online: 27 July 2011

Abstract This paper presents two parallel formulations for the Barnes-Hut algorithm on the Cell architecture, which differ in tree distribution and construction phases of the algorithm. In the initial parallelization, the domains are dynamically partitioned and assigned to the synergistic processing elements (SPEs), and SPEs construct local trees of the sub-domains in parallel. The enhanced parallelization scheme provides better clustering of the particles by sequentially constructing the global tree of the entire work space in the power processing element (PPE) and by partitioning the tree into sub-trees that can fit in the Local Store. SPEs operate on the sub-tree data and construct local trees in parallel. Our experimental evaluation indicates that this application performs much faster on the Cell BE compared to the Intel Xeon based system. Specifically, our first and second methods on the Cell BE outperform Intel Xeon by a factor of 5.8 and 7.1 for 8192 particles, respectively.

B. Demiroz · O. Tosun
Computer Engineering Department, Bogazici University, 34342 Istanbul, Turkey

B. Demiroz
e-mail: betul.demiroz@boun.edu.tr

O. Tosun
e-mail: tosuno@boun.edu.tr

H.R. Topcuoglu (✉)
Computer Engineering Department, Marmara University, 34722 Istanbul, Turkey
e-mail: haluk@marmara.edu.tr

M. Kandemir
Dept. of Computer Science and Engineering, Pennsylvania State University, University Park, PA 16802, USA
e-mail: kandemir@cse.psu.edu

Keywords N-body simulations · Barnes-Hut algorithm · Parallel algorithms · Cell BE architecture

1 Introduction

Most chip manufacturers [1, 2] are working on the design and optimization of chip multiprocessors (CMP), which are widely used in personal desktop computers and embedded systems, as well as large-scale parallel machines. A CMP is a collection of cores communicating through a cache hierarchy and the main design issue behind it is to improve performance, to limit power consumption and to increase the reliability of the architecture [3–8]. A CMP is a single chip containing multiple CPUs communicating through an interconnection fabric and using some memory components. When CMPs are compared to a single CPU based system, multiple CPUs are gathered together in a single chip, so each CPU can be operated using lower frequencies and supply voltages, which helps with both the power and reliability problems.

IBM Cell Architecture [1, 9, 10] is an example of heterogeneous multi-core architectures, jointly designed by IBM, Toshiba and Sony. The IBM Cell engine contains a traditional microprocessor called the power processing element (PPE), which deals with the orchestration and the coordination of the synergistic processing elements (SPEs). The tasks are performed by eight SPEs that work in parallel, and all components on the chip are connected via an element interconnect bus. The IBM Cell is suitable for applications demanding high performance, as it offers high computational power.

N-body is an important problem that can be applied to extensive applications from various domains in engineering and science, and it is one of the computation-intensive problems in the 13 dwarfs [11]. This problem simulates the motion of particles under pairwise interaction among n bodies for a predefined time period. At each time step, pairwise forces are calculated for all particles; this requires $O(n^2)$ computations, which are not feasible when millions of particles are considered. Therefore, in order to reduce the complexity of the problem, many approximation algorithms are proposed. The Barnes-Hut method [12] is one of the most popular approximation algorithms, which reduces the computational complexity of the N-body problem to $O(n \log n)$. It is widely used due to its simplicity and easily programmable nature without requiring complicated data structures.

This paper presents the design issues involved in mapping two different parallel implementations of the Barnes-Hut algorithm onto the Cell BE Architecture. In the initial parallelization, PPE partitions the domain into a set of sub-domains with an equal number of particles, and then it distributes a portion of sub-domains to SPEs and balances workload among SPEs. For every sub-domain, the local trees are constructed by all SPEs involved, and the force on each particle is computed using the local trees constructed by SPEs. This process continues until all sub-domains are processed. Finally, particles are distributed equally among SPEs, and each SPE updates the position and the velocity of the particles assigned to it. While constructing local trees, SPEs only consider the particles in the sub-domains; due to memory limitations, they do not have the cluster information of the upper levels of the tree representing the entire workspace. Due to this limitation, more nodes are explored in the force calculation phase, and the execution time of the method increases.

Our enhanced parallelization overcomes this problem by sequentially constructing the global tree representing the entire workspace in the PPE, as it can keep the entire tree in its memory at once, unlike SPEs. PPE divides the global tree into sub-trees, each of which contains the upper level nodes and stores them as contiguous data. Finally, it sends sub-tree data to the SPEs, which subsequently construct their sub-trees and calculate the force acting on all particles in the workspace.

Although Cell Architecture is suitable for computation-intensive parallel applications such as the Barnes-Hut algorithm, programming the Cell Architecture is quite difficult due to architecture-specific limitations and empirical optimization schemes. SPEs must communicate with the PPE in order to load the data they need, but this overhead can be negligible if the data to be used can be stored in contiguous data and if double-buffering is used. Cell Architecture has no branch prediction mechanism, so the algorithm should not be recursive, and loops should be minimized. The limitations of the Local Store (LS) should be also considered while designing code on SPEs. While designing our parallel methods, all of the above factors were taken into account. Based on our comparison study, the required extra effort is worthy. Specifically, our initial and enhanced parallelization methods achieve a speedup of 6.22 and 7.41, respectively, when results of 8 SPEs are compared to an Intel Xeon 3.0 GHz. processor using 4096 particles.

The rest of this paper is structured as follows. The next section summarizes the N-body problem and the Barnes-Hut method. The architectural features of the Cell processor are explained in Sect. 3. Section 4 presents the details of two parallel implementations of the Barnes-Hut algorithm and related empirical optimization schemes for the Cell processor. Then, we discuss the results of our experimental evaluation in Sect. 5. Finally, Sect. 6 summarizes our main conclusions.

2 Related work

The N-body problem is used to simulate the evolution of n particles in a space by calculating the pairwise forces among them. Due to the interactions among particles, a net force is exerted in each particle, which causes them to move within a specified time. In each time step, with the movement of the particles, the forces are recalculated, because the force is dependent on the distance between particles. Some application areas of the problem are astrophysics, molecular dynamics, plasma physics, embedded SAR data processing, and protein folding.

At each time step, the n-body problem requires n^2 interactions; it calculates all pairwise forces, so the exact formulation of this problem requires $O(n^2)$ computations. In order to reduce the complexity of the problem, many methods have been proposed. The Barnes-Hut method [12, 13], which has a computational complexity of $O(n \log n)$, is a popular algorithm due to its simplicity. Another approximation algorithm is the Fast Multipole Method [14] with a complexity of $O(n)$, which has working principals similar to those of the Barnes-Hut method except that it uses complex data structures.

The Barnes-Hut method employs a hierarchical tree representation of space and reduces the number of interactions among particles by grouping relatively close particles under a single tree node. The sequential implementation of this method has two phases: *the tree construction phase* and *the force computation phase*. In the tree construction phase, the domain is represented with a tree in which the leaves are the particles themselves, and internal nodes are the clusters grouping nearby bodies (i.e., particles). This phase starts with an empty space, and particles are added to the domain one by one. For a 2D space, the domain is recursively divided into four equal sub-domains if the domain contains more than one particle. This process is repeated until each sub-domain contains a single particle at each time step. The force acting on particles is calculated in the force computation phase. Instead of calculating pairwise interactions among the particles, the force between the nodes of the tree and particles is calculated using the distance between the particle and the center of mass of the node.

In order to cluster particles and represent the clusters with internal nodes, the center of mass and the total mass of the particles in the cluster are calculated, and this information is stored in each internal node. The net force on each particle is calculated by traversing all nodes of the tree, starting from the root. If the center of mass of an internal node is sufficiently far away from the particle, the particles contained in the internal node are approximated as a single particle, and the net force acting on the particle is calculated using internal nodes' mass and center of mass information. If the center of mass of an internal node is not sufficiently far away from the particle, then each of the sub-nodes of the internal node is explored and traversed recursively.

The multipole acceptance criteria (MAC) [12] determines whether a node is at a sufficient distance from a particle, and it is equal to the ratio of the dimension of the domain to the distance of the particle from the center of mass of the domain. If this ratio is less than the predefined constant λ, the node is at a sufficient distance from the particle and an interaction can be computed; otherwise, the node is expanded to its sub-nodes, and the force between the particle and center of mass of the sub-nodes is computed recursively. The speed and accuracy of the simulation is determined by setting a proper value for λ.

Although there are various parallel implementations of the Barnes-Hut method in the literature [13, 15], which are either for message-passing architectures [16] or shared-memory architectures [17], they cannot be directly mapped to the Cell architecture. The sequential non-recursive Barnes-Hut algorithm given in Fig. 1 is the basis of our parallel implementations running on the Cell architecture.

```
For time ← 0 to endTime do
    For i ← 0 to particleNumber do
        Insert particle i to the tree
        Update center of mass and total mass of each internal node on the way
    End
    For i ← 0 to particleNumber do
        Add root node to visitList
        While visitList is not empty
            Calculate distance between particle i and center of mass of node
            If center of mass of the node and particle is distant
                Calculate force acting on particle i
            Else
                Add children of node to visitList
            End
        End
    End
    For i ← 0 to particleNumber do
        Update position and velocity of particles
    End
End
```

Fig. 1 Sequential version of the Barnes-Hut algorithm

3 The Cell BE architecture

The Cell Broadband Engine [1, 9, 10] is a high-performance architecture designed by Sony, Toshiba, and IBM that targets multimedia and gaming applications. The Cell architecture is used in Sony's Play Station 3 gaming console, Mercury Computer System's dual Cell-based blade servers and IBM's Cell Blades (called QS20, QS21 and QS22). The Cell is designed to increase microprocessor efficiency in terms of both power and performance. The clock speed of the Cell processor is 3.2 GHz, and it has a single-precision peak performance of 204.8 Gflops/s and a double-precision peak performance of 14.6 Gflops/s.

A general overview of the Cell Broadband Engine Architecture is shown in Fig. 2. The Cell consists of a traditional microprocessor (power processing element—PPE), 8 smaller and simpler processors (synergistic processing elements—SPEs), and an element interconnect bus (EIB) that connects the processors and provides access to main memory and I/O devices. The PPE is the main processor in the Cell BE. It is a 64-bit PowerPC core with two levels of on-chip cache, 32 Kbyte L1 instruction and L1 data cache, and 512 Kbyte L2 cache.

Each SPE consists of a Synergistic Processor Unit (SPU), a Memory Flow Controller (MFC), and a Local Store (LS). The instruction set of SPEs is designed to take advantage of 128-bit registers, and most of the instructions are SIMD instructions. Memory operations access 128-bits at a time, even when the requested data are 8-bits, so the programmer should request 128-bits in each memory operation for an efficient implementation. SPEs are in-order processors with two instruction pipelines, namely the odd pipeline and the even pipeline. The odd pipeline handles memory and branch instructions, and the even pipeline is responsible for arithmetic operations. In a single clock cycle, SPEs can dispatch two instructions if these instructions have no data dependency. SPEs do not have a cache, but they have a 256 Kbyte Local Store (LS) which can be called private memory. Both the program and the data should be in LS to be executed. SPEs have no direct access to the main memory, and a DMA controller is used to perform high bandwidth data transfers among the local store, main memory and other local stores.

The Element Interconnect Bus (EIB) connects all components of the Cell processor including the PPE, the SPEs, the main memory, and I/O. It supports a peak bandwidth of 204.8 Gbytes/s [19]. EIB is constructed of four 16-byte wide unidirectional rings, two in each direction.

Many algorithms from a variety of domains have been parallelized and developed on the Cell architecture due to the performance and power utilization of the Cell processor. The potential of the Cell processor for several scientific computing kernels and the performance and power ef-

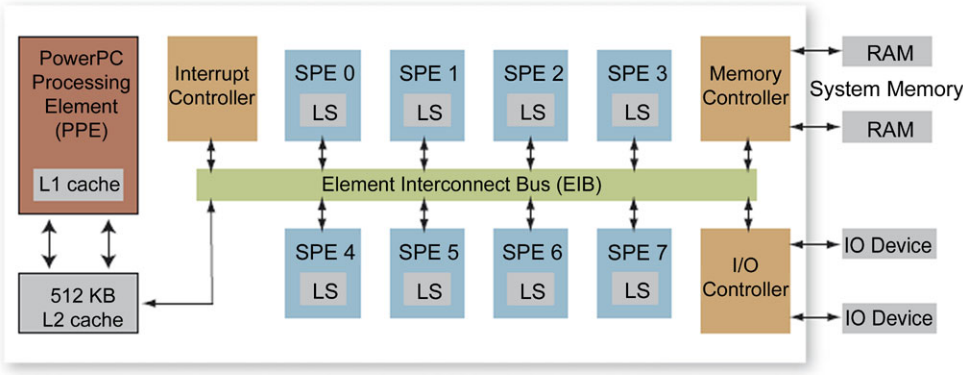

Fig. 2 IBM Cell broadband engine architecture [18]

ficiency of the Cell architecture for these kernels have been presented recently [20]. The parallel implementation of FFT on the Cell processor is also shown in the literature [21–23]. Additionally, a complexity model for algorithm designs on the Cell with an implementation of the list ranking problem by using the model is given in [24]. The performance and design choices of the breadth-first search algorithm are explored in [25]. Bio-informatics applications [26] have also been tested on the Cell processor. These studies highlight the performance and the architectural restrictions of the Cell processor for various applications.

4 Mapping and optimizing the Barnes-Hut method on the Cell processor

This section presents common characteristics of our two parallel implementations of the sequential Barnes-Hut Algorithm (given in Fig. 1) for the Cell Architecture. As part of parallelization, we aim to distribute data across SPEs. Since our workspace contains large number of particles and each SPE contains a 256 KB Local Store, it is not possible to hold the entire data and the tree on each SPE. Therefore, the tree representing a part of the domain is gradually constructed in the SPEs, and the force calculation is replicated for each local tree of the SPEs. In our initial parallelization, the number of particles in the domain is distributed equally among all SPEs in order to overcome both the limited size of the local stores and the time wasted on the synchronization barrier due to load imbalances among SPEs. The total number of nodes in the global tree is distributed among the SPEs in our enhanced parallelization.

In order to exploit the unique features of the Cell Architecture, a set of issues such as avoiding branches, vectorizing the corresponding code, and overlapping memory access with computation, should be considered [27]. Since the IBM Cell Engine has no branch prediction mechanism, we consider the non-recursive version of the Barnes-Hut algorithm as the starting point. Some of the if-then-else statements in the code are replaced with *select bits* instruction for eliminating branches. Additionally, the loops in the code are unrolled partially by decreasing the number of iterations and replicating the instructions in the loops. In order to reduce memory access latencies, DMA transfers and computations are overlapped through double buffering. The Barnes-Hut method performs the same computations on a large amount of 3D data repeating in each direction. In the force calculation phase (which is common in our two parallel implementations, as explained below), pairwise computations between particles and tree nodes are performed, where each computation is between a particle and the center of mass of a tree node. Each particle and the tree node are represented with a vector, and all operations performed are SIMDized. All data transferred between memory and the local store are stored contiguously in memory. Double-buffering usage is efficient for this type of implementation. After both the force calculation and the position and velocity updates are performed, the SPEs are synchronized using mailboxes that send and receive 32-bit messages.

```
Initialize Data Structures
Create SPEs
DMA: Inititate transfers to put blocks of particle coordinates to LS
Synchronization using mailbox
For time ← 0 to endTime do
    Arrange all particles in the sub-domains
    Use load information of the sub-domains and distribute load between SPEs
    Map sub-domains to SPEs
    Wait for SPEs to finish tree construction and force calculation
    Synchronization using mailbox
    Wait for SPEs to finish position and velocity updates
    Synchronization using mailbox
End
```

Fig. 3 View within PPE for the initial parallelization

4.1 Initial parallelization of the Barnes-Hut method

Our initial parallelization has three phases: *the domain decomposition phase*, *the tree construction and the force calculation phase*, and *the position and velocity updates phase*. The following subsections explain these phases in detail. The related pseudo-codes for the PPE-specific and SPE-specific parts are given in Fig. 3 and Fig. 4, respectively.

Fig. 4 View within SPE for the initial parallelization

```
DMA: Inititate transfers to obtain blocks of particle coordinates
For i ← 0 to particleNumber do
    Determine which sub-domain each particle belongs to
End
DMA: Put  the data with its sub-domain information from LS back to main memory
Synchronize using mailbox
For time ← 0 to endTime do
    While more sub-domains to visit
        While more blocks to process
            DMA: Load mass and coordinates of particles in the sub-domain from main memory into LS
            For i ← 0 to subParticles do
                Insert particle i to the tree
                Update center of mass and total mass of each internal node on the way
            End
        End
        While more blocks to process
            DMA: Load mass and coordinates of all particles in the space
            DMA: Load force of all particles in the space
            For i ← 0 to allParticles do
                Add root node to visitList
                While visitList is not empty
                    Calculate distance between particle i and the center of mass of the node
                    If center of mass of the node and particle is distant
                        Calculate force acting on particle i
                    Else
                        Add children of node to visitList
                    End
                End
            End
            DMA: Put force of all particles in the space
        End
        DMA: Put load information of sub-domain into MM
    End
    Synchronize using mailbox
    While more blocks to process
        For i ← 0 to particleNumber do
            DMA: Obtain force exerted by other SPEs on particles
            Calculate total force acting on particles
            Update position and velocity of particles
            Determine which sub-domain each particle belongs to
        End
        DMA: Put new position, sub-domain and velocity of particles from LS to main memory
    End
    Synchronize using mailbox
End
```

4.1.1 Domain decomposition

The domain is decomposed into smaller parts (called *sub-domains*) in which each part contains a limited number of particles that can fit in the LS of each SPE. Initially, the PPE fetches the positions, velocities, and masses of particles, and it distributes the position of particles equally to the SPEs to obtain their sub-domain information. After the PPE obtains this information from the SPEs, it constructs sub-domains and uses the load information of sub-domains to give equal workload to each SPE. In the first iteration, the workload is divided into w equal units of data, where total number of particles in the workload is equal to $w * p$, and p is the number of SPEs used in execution. When the first iteration ends, the number of nodes explored by the local tree representing a sub-domain during force calculation in the previous iteration is used as the workload metric of the sub-domain for the current iteration. After the PPE completes sub-domain allocation, it sends the sub-domains to the SPEs on which they will operate. Finally, each SPE fetches its data from memory and starts its local tree construction.

4.1.2 Tree construction and force calculation

In this phase, an octree representing the particles in a sub-domain is constructed by each SPE. Then, the force acting on all particles in the workload is calculated on all SPEs using their own local trees. Since the SPEs have local stores of limited size, they operate on many sub-domains by obtaining each sub-domain one by one. When the local tree

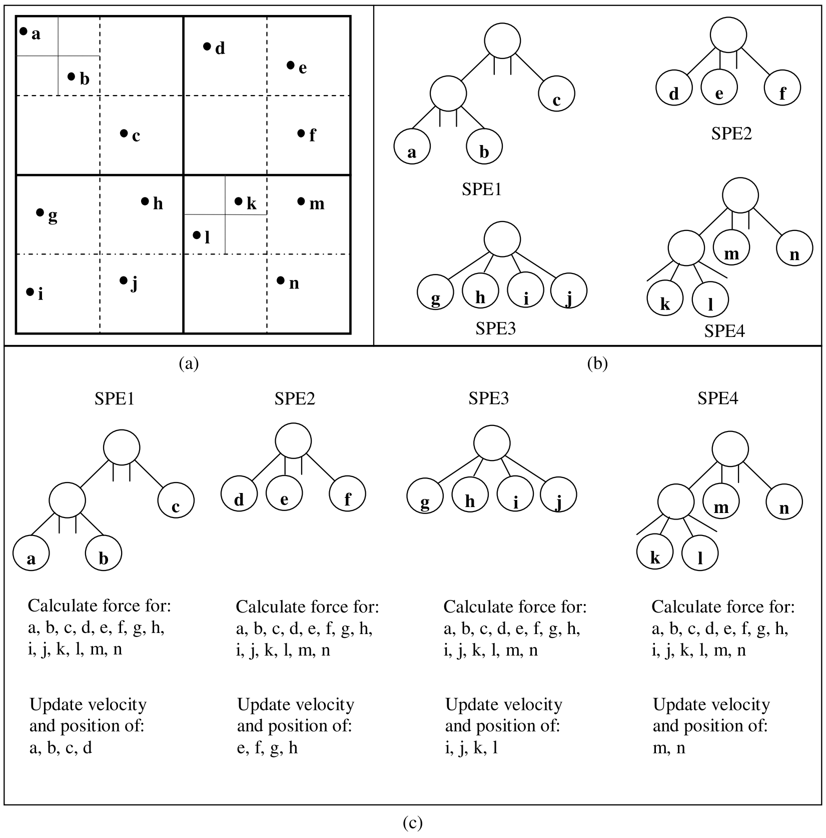
a
b
c
d
e
f
g
h
i
j
k
l
m
n
SPE1
SPE2
SPE3
SPE4
(a)
(b)
Calculate force for: a, b, c, d, e, f, g, h, i, j, k, l, m, n
Update velocity and position of: a, b, c, d
Update velocity and position of: e, f, g, h
Update velocity and position of: i, j, k, l
Update velocity and position of: m, n
(c)

Fig. 5 A typical step of the initial parallelization with 14 particles using 4 SPEs. (**a**) Domain decomposition (**b**) Local tree construction in SPEs (**c**) Force calculation and position updates in SPEs

representation of the sub-domain is finished, it starts the force calculation among the particles and the sub-nodes by traversing the tree starting from the root. Interactions with the top nodes of the tree are calculated first, and then the sub-nodes are explored, if necessary. If the center of mass of an internal node is sufficiently far away from the particle, the particles contained in the internal node are approximated as a single particle, and the net force acting on the particle is calculated using the internal nodes' mass and center of mass information. In order to determine whether a node is sufficiently far away from a particle, the multipole acceptance criteria [12] (MAC) is computed, which is the ratio of the dimension of the sub-domain to the distance of the particle from the center of mass of the given sub-domain. If a particle is too distant from a cluster of particles (where MAC is less than a predefined λ value), the particles contained in the cluster are represented with a single particle. Each SPE is responsible for different sub-domains, and there will be more interactions between particles and nodes that are close to each other in the domain. As soon as the force calculation ends, each SPE writes back the calculated force values of all particles in its local tree to the memory and obtains the data of the new sub-domain. The tree construction and force calculation phase continues until all sub-domains assigned to the SPEs are processed.

Figure 5(b) shows the local trees that are constructed by each SPE for the domain given in Fig. 5(a). There are 14 particles in this domain, which are processed by 4 SPEs. In the first phase, the domain information of particles is calculated, and each SPE is assigned an equal number of particles to work on. Therefore, SPE1, SPE2 and SPE3 receive 4 particles each, and SPE4 receives the remaining 2 particles. In the second phase, sub-domains are distributed among SPEs. There are 4 sub-domains, so each SPE handles one sub-domain. According to the number of particles in the sub-domains, SPE1 and SPE2 receive 3 particles, SPE3 and SPE4 receive 4 particles. Although SPE1 and SPE2 have the same number of particles, their tree construction times differ, as the depths of their trees are not the same.

When the number of the nodes used to construct the local trees in Fig. 5(b) are compared, SPE1 and SPE3 have 5 nodes, SPE2 has 4 nodes, and SPE4 has 6 nodes. Therefore, SPE2 has the lowest running time for tree construction, and SPE4 has the highest running time. It should be noted that SPE1 and SPE2 differ in terms of the running times for tree construction, although they have an equal number of particles in their sub-domains. In the force calculation phase, the local tree is explored for each particle, as shown in Fig. 5(c). The running time of this phase is much longer than that of the tree construction phase. Additionally, the running time of this phase may vary among the SPEs due to the different number of nodes to be explored by themselves, which results in an unbalanced load among SPEs.

4.1.3 Position and velocity updates

After the force calculation phase is completed, the net force acting on each particle is calculated using all force values calculated on all local trees. Then, the particle's velocity and position values are updated accordingly. This process is repeated for a predefined number of iterations, and at the end of the simulation, each particle has a new position and velocity value.

4.2 Enhanced parallelization of the Barnes-Hut method

In our initial parallelization, implementation of the tree construction phase by using only the particles in the subdomains causes loss of clusters of the domain. The tree construction phase is the heart of the Barnes-Hut algorithm. A good clustering of particles results in fewer computations for the force calculation phase. Therefore, our enhanced parallel implementation is built on the idea of obtaining better clustering of the particles in the domain. Although the global tree is required in order to achieve this goal, it cannot be stored in the LS of the SPEs due to memory limitations. Therefore, the first phase of our enhanced parallelization is *the sequential tree construction phase*, where PPE constructs the global tree by using all particles in the domain and divides the global tree into sub-trees. The second phase of the enhanced parallelization is *the local tree construction and force calculation phase*, where SPEs use sub-trees in order to construct their local trees. The last phase, *the position and velocity updates phase*, is the same as in the previous parallelization; therefore, it is not repeated in the following subsections. The related pseudo-codes are given in Fig. 6 and Fig. 7.

4.2.1 Global tree construction

In this phase, PPE constructs a global tree that contains all the particles in the domain, and then it divides the global tree into sub-trees by using depth first search. The number of sub-trees depends on the total number of nodes generated in the global tree and the number of SPEs in use. In this parallelization, the total number of nodes are distributed equally among SPEs to have better load balancing. These sub-trees are stored as contiguous data in order to have a predictable memory access pattern. Each element in the array represents a node in the tree and contains the center, the total mass, the total number of children nodes, and the dimensions of the domain represented by the node. After PPE completes the generation of sub-trees, SPEs fetch their data from the memory and start local calculations.

```
Initialize data structures
Generate global tree
Create sub-trees and store node information to tree array
Map sub-trees to SPEs
Create SPEs
For time ← 0 to endTime do
    Wait for SPEs to finish tree construction and force calculation
    Synchronization using mailbox
    Wait for SPEs to finish position and velocity updates
    Generate global tree
    Create sub-trees and store node information to tree array
    Map sub-trees to SPEs
    Synchronization using mailbox
End
```

Fig. 6 View within PPE for the enhanced parallelization

4.2.2 Tree construction and force calculation

In this phase of the algorithm, each SPE obtains the number of sub-trees assigned to it and loads the nodes of the trees from memory to its local store one by one. The tree construction phase is simpler in this parallel implementation because the positions of the nodes are not searched in the entire local tree. Instead, this information is calculated by the PPE and is stored in the tree array. One disadvantage of this method is that additional variables are needed to store upper levels of the tree nodes, and this reduces the available space reserved for keeping local trees. On the other hand, this method has two advantages. It decreases the number of comparisons in finding the position of the particles in the trees thus reduces tree construction time. In addition, the upper levels of the local trees that contain the clustering information of the domain are included in the local trees generated. This reduces the total execution time of the force calculation phase by decreasing the number of comparisons in this phase. After local trees are constructed, the tree is traversed starting from the root, and the force acting on each particle in the domain is calculated. It should be noted that the force calculation phase implementation is the same as described in Sect. 4.1.2. This phase continues until all local trees assigned to the SPEs are processed.

Figure 8(a) shows the global tree that is sequentially constructed by the PPE for the domain given in Fig. 5(a). The global tree contains 20 nodes (excluding the root node), and the root node is replicated in all local trees assigned to 4 SPEs. The number of nodes distributed is 24, and each SPE should obtain approximately 6 nodes. While constructing local trees, the PPE first considers keeping the clusters together; then, it takes the number of nodes into account. In this enhanced parallelization, the number of nodes is considered, which is different than the initial parallelization in which the number of particles is considered instead. This change aims to balance load among SPEs.

Figure 8(b) shows the local trees constructed by SPEs and the particles used in force calculation, as well as position and

Fig. 7 View within SPE for the enhanced parallelization

```
For time ← 0 to endTime do
    While more trees to generate
        While more blocks to process
            DMA: Load tree nodes information from main memory into LS
            For i ← 0 to nodeNumber do
                Insert node to the tree
            End
        End
        While more blocks to process
            DMA: Load mass and coordinates of all particles in the space
            DMA: Load force of all particles in the space
            For i ← 0 to particleNumber do
                Add root node to visitList
                While visitList is not empty
                    Calculate distance between particle i and the center of mass of the node
                    If center of mass of the node and particle is distant
                        Calculate force acting on particle i
                    Else
                        Add children of node to visitList
                    End
                End
            End
            DMA: Put force of all particles in the space
        End
    End
    While more blocks to process
        For i ← 0 to particleNumber do
            DMA: Get force exerted by other SPEs to particles
            Calculate total force acting on particles
            Update position and velocity of particles
        End
        DMA: Put new position and velocity of particles from LS to main memory
    End
    DMA: Put new position, sub-domain and coordinates of particles from LS to main memory
    Synchronization using mailbox
End
```

velocity update phases. The local tree includes upper level tree information in Fig. 8(b). For a higher number of particles, this information is very valuable, because it decreases the number of nodes explored.

5 Experimental work and analysis of results

The performance results presented in this section are from actual runs on an IBM Blade Center QS20 with two 3.2 GHz Cell BE processors, 512 KB Level 2 cache per processor and 1 GB memory. Our code is compiled using gcc compiler in the Cell SDK 3.1 with -O3 optimization flag for performance analysis. As mentioned before, loop unrolling, double-buffering, and vectorization are used to decrease execution time on the SPEs. The code running on the PPE is also optimized by loop unrolling, branch elimination, and vectorization. In all experiments performed, the number of particles used varies from 1024 to 32768, the number of buffers used is 2, λ is 0.5, and the number of iterations is 50, unless otherwise stated.

In the first set of tests, the performance of our algorithm is measured by varying the number of the SPEs used for different particle sizes. Figures 9 and 10 show the speedup values that are normalized to a single SPE for the initial and enhanced parallelizations, respectively. The running time of the PPE is also included in the experiments. Considering that the Barnes-Hut algorithm is branchy, the PPE runs the code faster than a single SPE does. When 1024 particles are considered, 2 SPEs run 1.61, 4 SPEs run 2.2, and 8 SPEs run 3.98 times faster than 1 SPE in the initial parallelization. Similarly, in the enhanced parallelization, 2 SPEs run 1.93, 4 SPEs run 3.4, and 8 SPEs run 6.17 times faster than 1 SPE.

For the case of 32768 particles, 2 SPEs complete 1.95, 4 SPEs complete 4.18, and 8 SPEs complete 8.45 times faster in our initial parallelization. A speedup of 1.98 for 2 SPEs, 3.86 for 4 SPEs, and 7.32 for 8 SPEs are achieved in the enhanced parallelization. This shows that the application

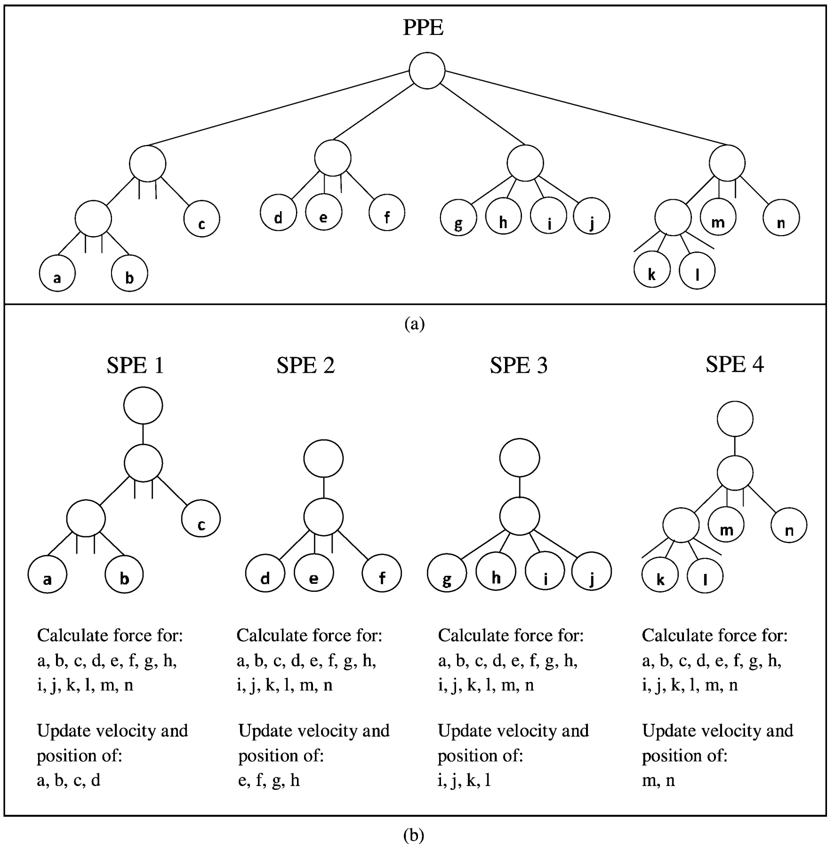
PPE
a
b
c
d
e
f
g
h
i
j
k
l
m
n
(a)
SPE 1
SPE 2
SPE 3
SPE 4
Calculate force for: a, b, c, d, e, f, g, h, i, j, k, l, m, n
Update velocity and position of: a, b, c, d
Calculate force for: a, b, c, d, e, f, g, h, i, j, k, l, m, n
Update velocity and position of: e, f, g, h
Calculate force for: a, b, c, d, e, f, g, h, i, j, k, l, m, n
Update velocity and position of: i, j, k, l
Calculate force for: a, b, c, d, e, f, g, h, i, j, k, l, m, n
Update velocity and position of: m, n
(b)

Fig. 8 A typical step of the enhanced parallelization with 14 particles using 4 SPEs. (**a**) Global tree construction (**b**) Local tree construction, force calculation and position updates in SPEs

scales well as the number of the SPEs increases in both implementations, and load balancing is better in the enhanced parallelization. The performance of the SPEs with respect to CPI values is also measured using the IBM Cell Simulator. For the initial implementation of the method, the CPI values of the tree construction phase range between 1.99 and 2.03, and in the force calculation phase, they are between 1.72 and 1.79. When the enhanced parallelization is considered, the CPI values of tree calculation and force calculation phases are 1.97 and 1.69, respectively. The percentage of total cycles stalled due to branch miss is equal to 25.7% for tree construction, and it is equal to 9.8% for force calculation. The CPI values are higher than the theoretical CPI value of 0.5 due to the branchy nature of the algorithm because in both phases, the tree is traversed for each particle.

The second set of experiments presents a performance comparison of our implementations with that of the Intel Xeon 3.0 GHz processor running the Linux operating system. The Barnes-Hut code on the Intel Xeon processor is compiled with gcc version 3.4.6 and -O5 optimization flag. The optimizations on the application code (except Cell specific optimizations including parallelization and vectorization) are applied to both architectures for a fair comparison. Additionally, the SPEs can construct trees with a limited number of particles because they have LS limitations, and this increases the number of nodes explored in the force

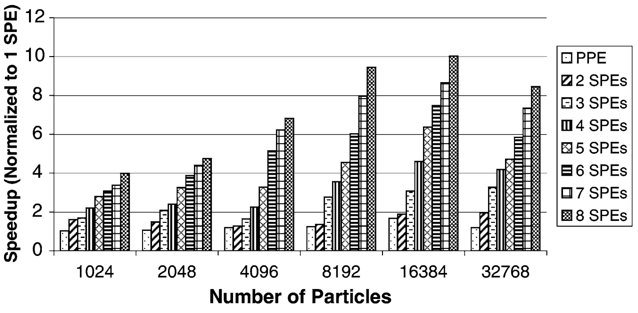

Fig. 9 Speedup with respect to different number of SPEs and PPE—initial parallelization

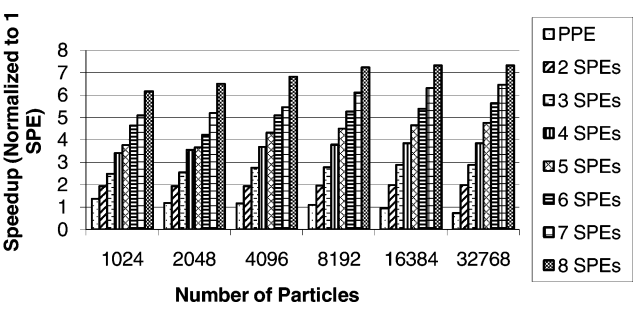

Fig. 10 Speedup with respect to different number of SPEs and PPE—enhanced parallelization

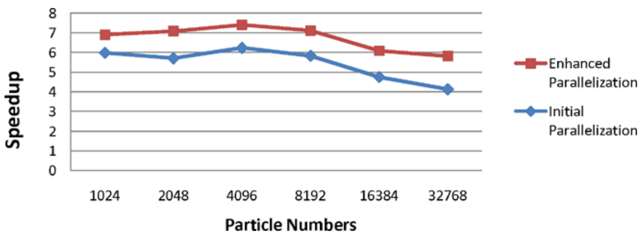

Fig. 11 Speedup of 8 SPEs relative to the Intel Xeon with respect to different number of particles

calculation phase, whereas Intel Xeon has no such memory limitations, so all particles are considered simultaneously in the tree construction phase.

Due to the 256 KB LS size, each SPE constructs a tree consisting of at most 370 nodes, and a single sub-domain used for tree generation contains at most 208 particles in each LS in the initial parallelization. To ease workload distribution and load balancing, the workspace is initially divided into 512 sub-domains in each tree generation step. Then, the SPEs are assigned N sub-domains, where the total number of particles in each sub-domain is less than or equal to 208. As the number of particles increases, all nearby bodies may not be clustered in a single tree because of space limitations, and this increases the number of nodes explored in the force calculation step. As explained in Sect. 4.2, the bottleneck of this method is solved by distributing tree nodes containing cluster information in the enhanced parallelization. This approach decreases the number of nodes explored in the force calculation phase and improves the most compute-intensive phase of the algorithm, resulting in a speedup of 1.4 for 32768 particles.

Figure 11 compares the speedup of the initial and enhanced parallelizations running on 8 SPEs with respect to Intel Xeon by varying the number of particles considered. When 4096 particles are considered, the Cell outperforms Intel Xeon with a speedup of 6.22 for our initial parallelization of the Barnes-Hut method, and it is equal to 7.4 for the enhanced parallelization. As the number of particles increases, we can no longer achieve an ideal speedup as there is an increase in the total number of local trees generated in a single step due to LS size restriction. The number of trees generated for 1024, 8192, and 32768 particles are 8, 64, and 200 respectively. As a result, all of the clusters of the domain cannot be stored in the LS; consequently, the number of clusters used in the force calculation phase decreases. When a lower number of clusters is considered, the number of nodes explored and the number of comparisons made increases, which decreases the performance of the system.

Figure 12 and Fig. 13 compare the running times of parallel implementations on the Cell processor for 1 SPE, 8 SPEs, PPE only, and the Intel Xeon cases by varying the number of particles in the domain. In Fig. 12, the best performance is obtained when 8 SPEs of the enhanced parallelization are

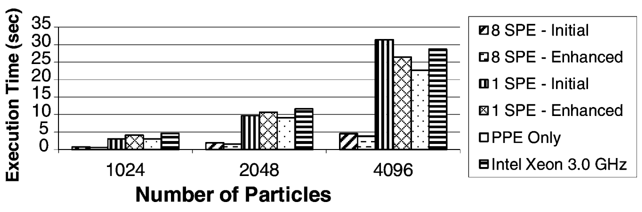

Fig. 12 Performance comparison with respect to number of particles given in the range [1024–4096]

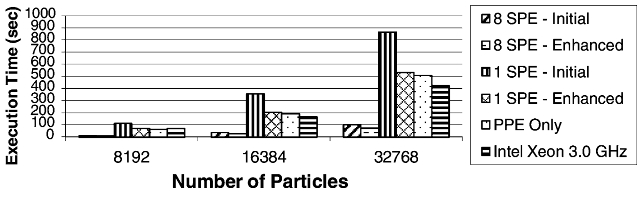

Fig. 13 Performance comparison with respect to number of particles given in the range [8192–32768]

used. For 1024 and 2048 particles, Intel Xeon shows the worst performance. For the case of 4096 particles, 1 SPE shows the worst performance and the PPE only case outperforms both Intel Xeon and 1 SPE. As the number of particles increases, the performance of Intel Xeon increases, and it outperforms both the PPE only and 1 SPE cases.

The average execution time of different phases of our enhanced parallelization for 8 SPEs is shown in Table 1, where the last two rows are for the overall computation and communication times. In this table, the "PPE-Specific" row denotes the time spent on global tree construction. As the number of particles in the workspace increases, the overhead of this phase compared to the total execution time decreases. Specifically, it takes 7.2% and 0.84% of the total execution time when 1024 and 32768 particles are considered. Similar behavior can be observed in the tree construction phase handled by SPEs; 4.4% and 0.54% of the total execution time are spent in this phase for 1024 and 32768 particles, respectively.

The update positions and velocities phase takes the least amount of time, as this phase includes SIMDized calculations and is straightforward. This phase provides good load balancing, so the synchronization barrier at the end of this phase takes only 0.1% of the total execution time for 32768 particles. The synchronization barrier at the end of the force calculation phase puts a higher overhead in running time, since loads are distributed to SPEs dynamically and it is harder to balance load in this phase. Specifically, 1.27% of the of the total execution time for 32768 particles is spent in this barrier. As the number of particles increases, nearly the entire computation of the algorithm is for the force calculation phase; namely this phase takes 88.3% to 98.9% of the total computation time of the method. The average communication to computation ratio for the given number of parti-

Table 1 Communication and computation times (in msecs) for various phases of the parallel Barnes-Hut algorithm

Phase considered		Number of particles					
		1024	2048	4096	8192	16384	32768
Tree construction (SPE)	Computation	1.3	3.2	7.8	22.1	43.5	75.5
	DMA	0.04	0.1	0.2	0.6	1.1	2.1
Force calculation (SPE)	Computation	18.3	68.1	198.3	953.6	3851.5	11573.9
	DMA	0.3	1.2	3.9	22.1	94.4	2854.7
Update position (SPE)	Computation	1.1	2.2	5.4	7.3	20.3	41.6
	DMA	0.05	0.1	0.2	0.3	1.2	25.5
PPE-specific	Computation	2.1	5.2	10.0	21.4	46.1	116.8
Synchronization (SPE)	Force	4.4	13.8	9.7	210.0	435.8	2052.7
	Update	1.7	2.2	3.2	5.1	9.7	26.2
Entire application	Computation	22.8	78.8	221.5	1004.5	3961.6	11807.9
	Communication	6.5	17.5	17.4	238.3	542.2	2369.0

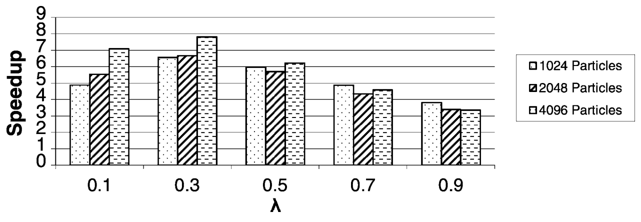

Fig. 14 Speedup of 8 SPEs relative to the Intel Xeon with respect to different λ values—initial parallelization

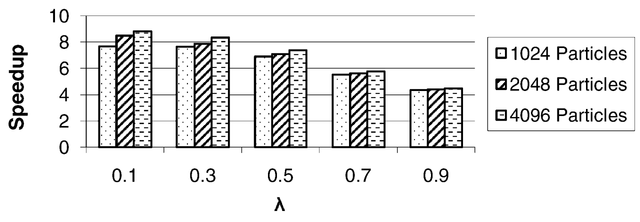

Fig. 15 Speedup of 8 SPEs relative to the Intel Xeon with respect to different λ values—enhanced parallelization

cles is equal to 0.19. Based on these results, the load balancing and communication of the application are acceptable.

As explained in Sect. 4.1.2, the MAC criteria calculates the ratio of the distance between the center of mass of the node and a particle to the dimension of the sub-domain. The term λ is used to decide whether to expand the corresponding node or to represent all particles that belong to the node with the center of mass of the node. The value of λ is in the given range $0 \leq \lambda \leq 1.0$; and if it is equal to 0, the Barnes-Hut application is turned into the N-Body problem in which all nodes are explored and the pairwise force among all particles is calculated. The increase in λ value decreases the number of nodes explored and also decreases the accuracy of the calculations performed. Since the application spends most of its time doing force calculations, the execution time decreases as the number of nodes explored decreases.

Figure 14 and Fig. 15 compare the performance of the Cell processor of 8 SPEs with that of the Intel Xeon for different λ values. When λ value is close to 1, the algorithm does not consider the individual positions and mass values of the particles in the force calculation phase; rather, it uses cluster information regarding the upper level nodes. When the SPEs do not contain the upper level cluster information due to space limitations, they consider lower level cluster information, which increases the number of comparisons made. As a result, the speedup decreases when λ is 0.7 and 0.9. When the λ value is close to 0, cluster information of the lower level nodes is more frequently used, which is mostly available in the local trees of the SPEs. Therefore, the performance of the application increases. A test of 4096 particles with a value of 0.3 for λ outperforms other alternatives by providing a speedup of 7.83 for the initial parallelization and a speedup of 8.85 for the enhanced parallelization.

In the last set of experiments, the effect of compiler optimization is measured for 1, 4, and 8 SPEs for a variable number of particles, as shown in Table 2. Here, compiler flag -O1 is used to reduce the code size and execution time. The compiler increases code performance without compromising on code size by applying the -02 compilation flag. Additionally, the highest level of optimization is achieved using the -03 compiler flag, which is our standard choice in the previous experiments. The Barnes-Hut algorithm is branchy and with the usage of the -O1 flag, the compiler tries to optimize loops and if statements. It rearranges program code and minimizes branches. The Cell architecture has no branch mechanism, so the best performance improvement is achieved with this compiler flag. When compiler flag -O1 is used, a speedup of 1.75 is obtained; for compiler flags -O2

Table 2 Execution time (in seconds) of Barnes-Hut method with compiler optimization flags for different number of SPEs and particles

Number of SPEs	Number of particles	Compiler optimization flag None	O1	O2	O3
1	1024	6.70	4.34	4.18	4.18
	2048	17.36	11.15	10.74	10.73
	4096	42.96	27.47	26.51	26.49
	8192	119.03	73.30	71.44	71.41
	16384	347.22	206.64	203.05	202.99
	32768	925.33	539.51	533.34	533.18
4	1024	2.04	1.27	1.23	1.23
	2048	5.01	3.12	3.01	3.01
	4096	12.00	7.45	7.20	7.18
	8192	32.03	19.35	18.89	18.85
	16384	91.62	53.68	52.79	52.71
	32768	242.61	139.65	138.22	138.06
8	1024	1.18	0.70	0.69	0.68
	2048	2.86	1.72	1.66	1.65
	4096	6.69	4.02	3.90	3.88
	8192	17.19	10.14	9.89	9.86
	16384	49.01	28.26	27.82	27.71
	32768	129.28	73.71	72.96	72.76

and -O3, a speedup of 1.77 is achieved, when 8 SPEs and 32768 particles are considered.

6 Conclusions

In this paper, we present and evaluate two parallel implementations of the Barnes-Hut method on the Cell architecture. The memory requirement of this application is very high, since both the local tree representing the domain and the positions and masses of the particles should be stored in the Local Stores of the SPEs simultaneously. Consequently, an efficient workload distribution on the SPEs is needed. In our initial parallelization, the workloads of the SPEs are the domains; and they are divided into sub-domains and are assigned to SPEs to overcome memory needs and to balance load. This method contains three phases. In the first phase, the domain is decomposed into smaller sub-domains and the sub-domains are assigned to the SPEs. In the second phase, tree construction and force calculation in each sub-domain are performed for sub-domains. Finally the velocities and positions of the particles are updated. In this approach, only the particles in the assigned sub-domains are used in the tree construction phase, so they do not contain many important clusters that are stored in the higher levels of the tree.

In our enhanced parallelization, a global tree representing the entire workspace is constructed by the PPE, and nodes including all clusters of the workspace are assigned to SPEs. This increases the effort in the tree construction phase, but decreases the execution time of the most time consuming phase, namely force calculation, significantly. After the tree construction phase is finished, the algorithm follows the same steps as in the initial parallelization. While programming the Cell architecture is more difficult than that of Intel Xeon (due to explicit on-chip memory management), our results show that the required extra effort is worthy. As part of the experimental study, the performance of the two proposed parallelizations are measured by using different number of SPEs, different λ values, and different number of particles. Our proposed parallelizations significantly outperform the sequential version running on Intel Xeon processor for the various test cases considered.

Acknowledgements This research was supported by The Scientific and Technological Research Council of Turkey (TUBITAK) with a research grant (Project Number: 108E035). Additionally, part of this research has been funded by Bogazici University Research Fund—08HA101D. Dr. Mahmut Kandemir is supported in part by NSF grants 1017882, 0963839, 720645 and a grant from Microsoft. The authors acknowledge Georgia Institute of Technology, its Sony-Toshiba-IBM Center of Competence, and the National Science Foundation, for the use of Cell Broadband Engine resources that have contributed to this research. A preliminary work regarding this research [28] was presented at the Workshop on Programmability Issues for Multicore Computers (MULTIPROG) at the *HIPEAC 2010 Conference*, Pisa, Italy.

References

1. Kahle, J., Day, M., Hofstee, H., Johns, C., Maeurer, T., Shippy, D.: Introduction to the Cell multiprocessor. IBM J. Res. Dev. **49**(4–5), 589–604 (2005)
2. Kongetira, P., Aingaran, K., Olukotun, K.: Niagara: a 32-way multithreaded SPARC processor. IEEE Microw. Mag. **25**(2), 21–29 (2005)
3. Olukotun, K., Hammond, L.: The future of microprocessors. ACM Queue **3**(7), 26–29 (2005)
4. Kumar, R., Tullsen, D., Jouppi, N.: Heterogeneous chip multiprocessors. IEEE Comput. Soc. **38**(11), 32–38 (2005)
5. Taylor, M.B., Kim, J., Miller, J., Wentzlaff, D., Ghodrat, F., Greenwald, B., Hoffmann, H., Johnson, P., Lee, J.-W., Lee, W., Ma, A., Saraf, A., Seneski, M., Shnidman, N., Strumpen, V., Frank, M., Amarasinghe, S., Agarwal, A.: The RAW microprocessor: a computational fabric for software circuits and general purpose programs. IEEE MICRO **22**(2), 25–35 (2002)
6. Sankaralingam, K., Nagarajan, R., Liu, H., Kim, C., Huh, J., Burger, D., Keckler, S.W., Moore, C.R.: Exploiting ILP, TLP, and DLP with the polymorphous TRIPS architecture. In: Proceedings of the 30th Annual International Symposium on Computer Architecture, pp. 442–433 (2003)
7. Swanson, S., Michelson, K., Schwerin, A., Oskin, M.: Wavescalar. In: Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture, Washington, DC, pp. 291–302 (2003)
8. Mai, K., Paaske, T., Jayasena, N., Ho, R., Dally, W.J., Horowitz, M.: Smart memories: a modular reconfigurable architecture. In: Proceedings of the Annual International Symposium on Computer Architecture, New York, NY, USA, pp. 161–171 (2000)

9. Hofstee, H.P.: Power efficient processor architecture and the Cell processor. In: Proceedings of the International Symposium on High Performance Computer Architecture, pp. 258–262 (2005)
10. Pham, D., Asano, S., Bolliger, M., Day, M.N., Hofstee, H.P., Johns, C., Kahle, J., Kameyama, A., Keaty, J., Masubuchi, Y., Riley, M., Shippy, D., Stasiak, D., Suzuoki, M., Wang, M., Warnock, J., Weitzel, S., Wendel, D., Yamazaki, T., Yazawa, K.: The design and implementation of a first-generation Cell processor. In: International Solid State Circuits Conference, San Fransisco (2005)
11. Asanovic, K., Bodik, R., Catanzaro, B., Gebis, J., Husbands, P., Keutzer, K., Patterson, D., Plishker, W., Shalf, J., Williams, S., Yelik, K.: The landscape of parallel computing research: a view from Berkley. Technical Report, EECS Department, University of California at Berkley, UCB/EECS-2006-183 (2006)
12. Barnes, J., Hut, P.: A hierarchical $O(n \log n)$ force calculation algorithm. Nature **324**, 446–449 (1986)
13. Grama, A., Kumar, V., Sameh, A.: Scalable parallel formulations of the Barnes-Hut method for N-Body simulations. In: Proceedings of Supercomputing, pp. 439–448 (1994)
14. Greengard, L.: The Rapid Evolution of Potential Fields in Particle Systems. The MIT Press, Cambridge (1988)
15. Ramachandran, U., et al.: Architectural mechanisms for explicit communication in shared memory multiprocessors. In: Proceedings of Supercomputing (1995)
16. Warren, M., Salmon, J.: A parallel hashed oct tree n-body algorithm. In: Proceedings of Supercomputing Conference (1993)
17. Singh, J., Holt, C., Totsuka, T., Gupta, A., Hennessy, J.: Load balancing and data locality in hierarchical n-body methods. J. Parallel Distrib. Comput. (1994)
18. Bader, D.A., Agarwal, V., Madduri, K., Kang, S.: High performance combinatorial algorithm design on the Cell broadband engine processor. Parallel Comput. **33**(10–11), 720–740 (2007)
19. Kistler, M., Perrone, M., Petrini, F.: Cell multiprocessor communication network: built for speed. Communication build for speed. IEEE MICRO **26**(3), 10–23 (2006)
20. Williams, S., Shalf, J., Oliker, L., Kamil, S., Husbands, P., Yelick, K.: The potential of the cell processor for scientific computing. In: Proceedings of the 3rd Conference on Computing Frontiers, pp. 9–20 (2006)
21. Bader, D.A., Agarwal, V.: FFTC: fastest Fourier transform for the IBM Cell broadband. In: Engine Eleventh Annual High Performance Embedded Computing Workshop (HPEC), Lexington, MA (2007)
22. Chow, A.C., Fossum, G.C., Brokenshire, D.A.: A programming example: large FFT on the Cell broadband engine. In: Tec. Conf. Proc. of the Global Signal Processing Expo (GSPx) (2005)
23. Cico, L., Cooper, R., Greene, J.: Performance and programability of the IBM/Sony/Toshiba Cell broadband engine processor. White paper (2006)
24. Bader, D.A., Agarwal, V., Madduri, K.: On design and analysis of irregular algorithms on the Cell processor: a case study of list ranking. In: The IEEE International Parallel and Distributed Processing Symposium, Long Beach, CA (2007)
25. Villa, O., Scarpazza, D.P., Petrini, F., Peinador, J.F.: Challenges in mapping graph exploration algorithms on advanced multi-Core processors. In: Parallel and Distributed Processing Symposium, Long Beach, CA (2007)
26. Sachdeva, V., Kistler, M., Speight, E., Tzeng, T.H.K.: Exploring the viability of the Cell broadband engine for bioinformatics applications. In: Parallel and Distributed Processing Symposium, Long Beach, CA (2007)
27. Brokenshire, D.A.: Maximizing the power of the cell broadband engine processor: 25 tips to optimal application performance. In: IBM Developer Works (2006)
28. Demiroz, B., Topcuoglu, H., Kandemir, M., Tosun, O.: Parallelizing Barnes-Hut method on the Cell BE architecture. In: HIPEAC 2010, Third Workshop on Programmability Issues for Multicore Computers (MULTIPROG'10), Pisa, Italy (2010)

Betul Demiroz is a research assistant in the Computer Engineering Department at Marmara University, Istanbul, Turkey. Her main research interests include task scheduling and mapping, parallel computing, chip multiprocessing. She received the B.Sc. and M.Sc. degrees in computer engineering from Marmara University, Istanbul, Turkey, in 2002 and 2004, respectively. She received the Ph.D. degree in computer engineering from Bogazici University, Istanbul, Turkey in 2011.

Haluk R. Topcuoglu is currently a Professor in Computer Engineering Department at Marmara University, Turkey. He received the B.S. degree and the M.S. degree in Computer Engineering from Bogazici University, Istanbul, Turkey in 1991 and 1993, respectively. He received the Ph.D. degree in Computer Science from Syracuse University, Syracuse, NY in 1999. His research interests mainly include task scheduling and mapping for multicore architectures, multithreading, parallel programming, and evolutionary algorithms for solving real world problems. He is a member of the IEEE, the IEEE Computer Society and the ACM.

Mahmut Kandemir is a professor in the Computer Science and Engineering Department at the Pennsylvania State University. He is a member of the Microsystems Design Lab.Dr. Kandemir's research interests are in optimizing compilers, runtime systems, embedded systems, I/O and high performance storage, and power-aware computing. He is the author of more than 80 journal publications and over 300 conference/workshop papers in these areas. He has graduated 11 Ph.D. and 8 masters students so far, and is currently supervising 15 Ph.D. students and 1 masters student. He has served in the program committees of 40 conferences and workshops. His research is funded by NSF, DARPA, and SRC. He is a recipient of NSF Career Award and the Penn State Engineering Society Outstanding Research Award. He currently serves as the Graduate Coordinator of the Computer Science and Engineering Department at Penn State.

Oguz Tosun is currently a Professor in the Computer Engineering Department at Bogazici University. He received his B.Sc. and M.Sc. degrees in EE from Istanbul Technical University in 1971. Later in 1977 he received his Ph.D. in Computer and Information Sciences from Syracuse University. He taught courses at SUNY at Oswego from 1974 to 1978 and at University of Santa Clara in 1981. His current research interests include high performance computational architectures, Fault-tolerant computers and cognitive robotics.