Computer Physics Communications 183 (2012) 1937–1946

Contents lists available at SciVerse ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

VLBI-resolution radio-map algorithms: Performance analysis of different levels of data-sharing on multi-socket, multi-core architectures

S. Tabik [a,*], L.F. Romero [a], P. Mimica [b], O. Plata [a], E.L. Zapata [a]

[a] Dept. Computer Architecture, University of Málaga, 29080 Málaga, Spain
[b] Department of Astronomy and Astrophysics, University of Valencia, Valencia, Spain

ARTICLE INFO

Article history:
Received 27 October 2011
Received in revised form
17 April 2012
Accepted 20 April 2012
Available online 27 April 2012

Keywords:
VLBI-resolution radio-map algorithms
Parallel approaches
Multi-socket multi-core systems
UMA
NUMA
NUCA

ABSTRACT

A broad area in astronomy focuses on simulating extragalactic objects based on Very Long Baseline Interferometry (VLBI) radio-maps. Several algorithms in this scope simulate what would be the observed radio-maps if emitted from a predefined extragalactic object. This work analyzes the performance and scaling of this kind of algorithms on multi-socket, multi-core architectures. In particular, we evaluate a sharing approach, a privatizing approach and a hybrid approach on systems with complex memory hierarchy that includes shared Last Level Cache (LLC). In addition, we investigate which manual processes can be systematized and then automated in future works. The experiments show that the data-privatizing model scales efficiently on medium scale multi-socket, multi-core systems (up to 48 cores) while regardless of algorithmic and scheduling optimizations, the sharing approach is unable to reach acceptable scalability on more than one socket. However, the hybrid model with a specific level of data-sharing provides the best scalability over all used multi-socket, multi-core systems.

1. Background and motivation

Most current high-performance computing systems are clusters. The compute nodes in these systems are usually multi-socket, multi-core commodity servers and blades. Typically, each node has two or four processor chips and each chip has two, four, six or eight cores; in addition, it is common for two, four or eight cores to share the Last Level Cache (LLC), usually L2 or L3. Furthermore, some chip manufacturers introduce Simultaneous Multi-Threading (SMT) capabilities into their processing units. As a result, each node is a shared-memory multiprocessor, cache coherent Non-Uniform Cache Access (NUCA) system combining SMT and Chip Multi-Processor (CMP) concurrency technologies. This configuration introduces different levels of sharing in the memory hierarchy (cache hierarchy is partially shared among the cores), resulting in non-uniform data sharing overheads.

Designing efficient parallel scientific applications from scratch, able to make the most of these architectures still remains a hard task. A significant number of recent works, such as those found in [1–3], explore how to best use these technologies by analyzing specific optimizations using toy multi-threaded benchmarks, on small core count multi-core nodes of up to eight cores or, on multi-core simulators. However, there exist quite a few related works in the literature that cover the whole parallelization process, from selecting the programming model that fits best in the application to finding the execution configuration that gives the best performance. We believe that application scientists still need to know about more new experiences of real applications on real machines.

This work evaluates three different parallel approaches on three multi-socket multi-core platforms of a state-of-the-art VLBI-resolution radio-map computation algorithm, called SPEV (SPectral EVolution). This Fortran multi-dimensional time-dependent hydrodynamic code is widely used in astronomy to understand the shape and behavior of active galaxies [4–10]. In particular, this work analyzes three parallel programming paradigms with different sharing levels, (1) a sharing programming approach that allows all participating threads access the shared data-structures along the simulation, (2) a privatizing programming approach based on data partition and replication of shared arrays and, (3) a hybrid approach that mixes both sharing and privatizing models. The analysis of performance and scaling has been carried out on three multi-socket, multi-core architectures: (i) a dual-socket quad-core Intel X5355 (Clovertown), (ii) a quad-socket eight-core Intel X7550 (Beckton) and, (iii) a quad-socket 12-core AMD Opteron 6172 (Magny-Cours). We determine the performance bottlenecks of each implementation, apply some optimizations to alleviate them and, find out manual parallelizing processes that can be systematized and then automated in future works.

* Corresponding author. Tel.: +34 95 34169; fax: +34 95 2132790.
E-mail address: stabik@uma.es (S. Tabik).

0010-4655/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.cpc.2012.04.016

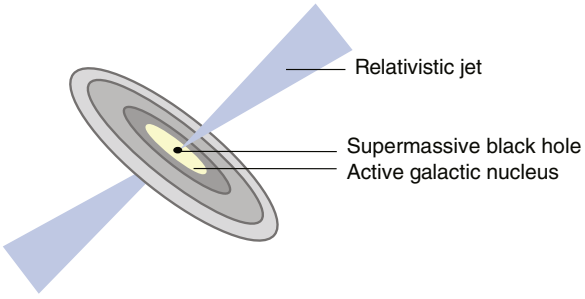

Fig. 1. The structure of an active galaxy.

2. Spectral evolution simulation

Before introducing the SPectral EVolution (SPEV) simulation, let us first define the terms active galactic nucleus (AGN), relativistic jets and, VLBI-maps. An AGN is a very compact region, of an average size of a couple of light-years, in the center of a galaxy which is characterized by the emission of enormous quantities of energy. Typically, the luminosity of an AGN is thousands of times larger than the luminosity of a normal galaxy. A simplified plot of the structure of an active galaxy is shown in Fig. 1. The supermassive black hole in the galaxy center accretes matter from an accretion disk which surrounds it. The relativistic (i.e., moving with velocities close to the speed of light) jets can be accelerated from a very magnetized and a very fast rotating disk. They remain collimated and relativistic up to distances of thousands of light years in some cases. See, e.g., [11,12] for a review of jet phenomenology and theoretical models.

Unfortunately, very little is directly known about the innermost regions of AGNs, since the structures at scales below light-year cannot be resolved at our present level of technology. However, we know that at intermediate scales (1–100 light years) jets emit synchrotron radiation which can be observed using radio telescopes on Earth. These scales are still too small for a single telescope, so that arrays of radio telescopes have to be used to resolve them. This is most commonly done using the Very Long Baseline Interferometry (VLBI) technique, i.e., simultaneous observations performed by several telescopes can be combined to provide observations which are to a certain extent equivalent to that made from a much larger telescope. The equivalent diameter of the hypothetical large telescope is comparable to the maximum separation between the telescopes in the array. In other words, VLBI has an advantage over other radio techniques in that the large distance between the telescopes allows for very long observational baselines, since telescopes can be located on different continents. By analyzing sequences of VLBI-radio maps taken at different epochs we hope to be able to understand the nature and behavior of relativistic jets and AGNs.

The observed radio maps of jets on the scales of light-years are not direct maps of their physical states, i.e., we cannot directly observe the density, pressure, velocity and, magnetic field of the emitting plasma. The emission structure is greatly modified by the fact that a distant (Earth) observer detects the radiation emitted from a jet which moves at relativistic speed and forms a certain angle with respect to the line of sight. Time delays between different emitting regions, Doppler boosting and light aberration shape decisively the observed aspect of every time-dependent process in the jet. The observed patterns are also influenced by the travel path of the emitted radiation towards the observer since Faraday rotation and, most importantly, opacity modulate total intensity and polarization radio maps. Moreover, there are other effects that can be very important for shaping VLBI observations which do not unambiguously depend on the hydrodynamic jet structure, namely, radiative losses, particle acceleration at shocks, pair formation, and so forth. Therefore, simulations needs to be shaped by considering all these kinds of distortions.

2.1. Description of SPEV-algorithm

SPEV [4] is a multi-dimensional time-dependent hydrodynamic simulation developed to study the evolution of radio components in relativistic jets. In fact, the kernel of SPEV solves the problems encountered by most other radio-map computing algorithms in astrophysics, such as those found in [5–10,13]. The key idea of SPEV is to operate on a pre-computed temporal sequence of (magneto) hydrodynamic states (i.e., velocity, density, pressure, magnetic field) of an extragalactic jet simulation and to compute the jet radio emission as it would be observed on Earth. This is done by evolving a set of Lagrangian particles whose emission is observed (i.e., calculated) at a Virtual Detector (VD). These Lagrangian particles represent groups of the real particles (usually electrons) which are assumed to be responsible for the observed radiation. Each Lagrangian particle contains information on the electron energy distribution (crucial for the emission calculation).

After creating and initializing a number of families of relativistic particles, the algorithm iteratively applies three phases: *evolve*, *inject* and *emit* on the jet of particles.

- *Evolve* computes the evolution in a two-dimensional non-uniform axisymmetric distribution of N_{spec} families of particles from one state to the next one.[1] It calculates the current state of each single particle, including its position and velocity, as well as the current state of the jet fluid at the particle location. The position $(x(t), y(t))$ of the particle follows these equations:

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = v(x, y, t)$$
$$\frac{\mathrm{d}y(t)}{\mathrm{d}t} = v(x, y, t)$$

  where t is the time of the snapshot currently being processed and v is the fluid velocity at (x, y). Furthermore, it evolves the electron energy distribution within a particle by solving the ensemble-averaged Boltzmann equation [14–16].

$$p^\beta \left(\frac{\partial f}{\partial x^\beta} - \Gamma^\alpha_{\beta\gamma} p^\gamma \frac{\partial f}{\partial p^\alpha} \right) = \left(\frac{df}{d\tau} \right)_{\text{coll}} \quad (1)$$

$$p^\beta \left(\frac{\partial f}{\partial y^\beta} - \Gamma^\alpha_{\beta\gamma} p^\gamma \frac{\partial f}{\partial p^\alpha} \right) = \left(\frac{df}{d\tau} \right)_{\text{coll}}. \quad (2)$$

  In this equation f is a function of x^α, y^α and the components of the 4-momentum p^α; $\Gamma^\alpha_{\beta\gamma}$ are the Christoffel symbols, the right hand side represents the collision term and, τ is the particle proper time. The particles and their evolution are stored in multi-dimensional matrices. Particles that go out of the fluid-domain are eliminated from the simulation since the information about the area outside is not available.
- *Inject* periodically introduces new particles in order to conserve the stability of the system. The new calculated data are added to the state matrices.
- *Emit* computes the observed emission by the VD. The first step is a conversion from the 2D into a 3D particle distribution by performing rotation around the jet axis of all particles such that each particle becomes a torus with a radius r, where r is the original distance from the axis. In practice this torus is represented by a set of particles "replicas" occupying most of its

[1] Each particle family corresponds to those Lagrangian particles which have been injected into the jet at a fixed distance from the jet axis, and from there they are transported by the jet fluid. The physical size of each particle depends on N_{spec}, i.e., for larger N_{spec} particles become smaller. Since particles eventually have to fill the whole jet volume, increased N_{spec} has as a consequence an increase in the total number of particles. The dependence of the total number of particles is $O({N_{spec}}^2)$.

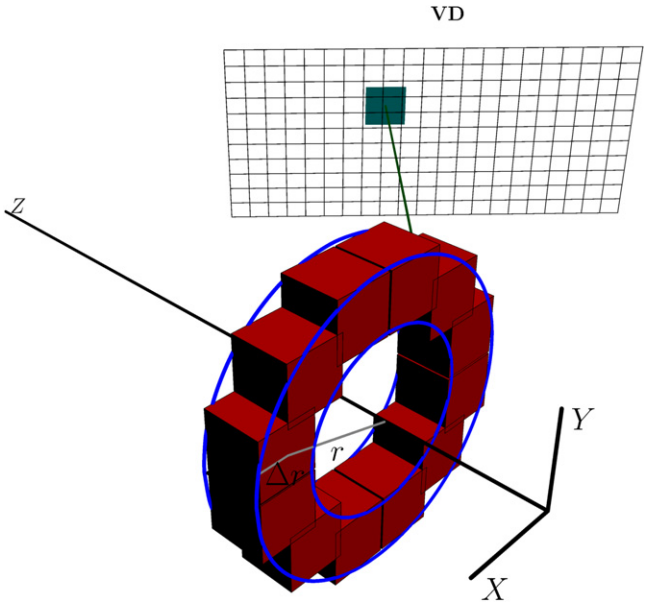

Fig. 2. An illustration of emit stage: a two-dimensional particle whose center is located at a distance r from the jet axis and whose spatial extent, Δr, is replicated as a number of cubes (shown in red) with the intent of approximating a torus whose outlines are plotted in blue. A virtual detector (VD) is located far away from the jet to capture the projection of each particle's emission onto its plane. An example of a particle image in VD is shown in dark green. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

volume. See Fig. 2 for illustration. Once the replicas have been created, their frequency dependent emission and absorption coefficients j_ν and κ_ν are computed:

$$j_\nu = \frac{\sqrt{3}e^3 B\delta^2}{4\pi m_e c^2}\int_{\gamma_{\min}}^{\gamma_{\max}} \mathrm{d}\gamma\; n(\gamma) F\left(\frac{\nu}{\delta\nu_0\gamma^2}\right)$$

$$\kappa_\nu = \frac{\sqrt{3}e^3 B}{8\pi m_e^2 c^2 \delta\nu^2}\int_{\gamma_{\min}}^{\gamma_{\max}} \mathrm{d}\gamma\; \gamma^2\left[-\frac{\mathrm{d}}{\mathrm{d}\gamma}\left(\frac{n(\gamma)}{\gamma^2}\right)\right] F\left(\frac{\nu}{\delta\nu_0\gamma^2}\right)$$

where e and m_e are the electron charge and mass, c is the speed of light, B is the magnetic field at the replica position and $\delta = [\Gamma(1-\beta\cos\theta)]^{-1}$ is the Doppler factor, which takes into account for the relativistic beaming and Doppler shift. $c\beta\cos\theta$ is the projection of the replica velocity vector along the line of sight and $\Gamma = (1-\beta^2)^{-1/2}$ is the Lorentz factor. The constant $\nu_0 = 3eB/4\pi m_e c$ is the gyrofrequency. The function $F(x)$ is defined as

$$F(x) := x\int_x^\infty \mathrm{d}\xi K_{5/3}(\xi)$$

where $K_{5/3}(x)$ is the modified Bessel function of index $5/3$. $n(\gamma)$ is the number density of ultra-relativistic electrons and is the first momentum of the distribution function f evolved using the Eq. (2). As the angle of a replica in a relativistically moving flow determines the Doppler shift, each of the replicas produces a slightly different emission and absorption coefficient. The pixels of the VD to which each replica contributes are determined as shown in dark green in Fig. 2. The contributions from all replicas at all times which contribute to a particular pixel are recorded separately, although it may occur that more than one particle contributes to the same pixel at the same moment. Finally, the value of each pixel is evaluated at the end of the calculation by solving the radiative transfer equation:

$$\frac{\mathrm{d}I_\nu}{\mathrm{d}s} = j_\nu - \alpha_\nu I_\nu$$

where s is the distance along the line of sight.

In summary SPEV serial code can be sketched as shown below:

```
!create "NumParticles" particles
call create(N_spec)
do curFile = startFile, endFile
  call readHydrodynamicData(curFile)
  do curFileBlock = 1, NumFileBlock
    curParticle = NumParticles
    !Evolve phase
    dowhile(curParticle .le. NumParticles)
      call computeTrajectory(curParticle)
      if(curParticle is inside the grid) then
        call evolveEnergy(curParticle)
      else
        call delete(curParticle)
        NumParticles=NumParticles-1
      endif
      curParticle=curParticle-1
    enddo
    !Inject phase
    dowhile( mod(Δ t, Dt) .eq. 0){
      do curParticle=1, NumParticles_to_be_injected
        call evolve( curParticle)
      enddo
    enddo
    !Emit phase
    do curParticle= 1, NumParticles
      call computeNumReplicas(curParticle)
      !For each replica, find the border of the VD 3D-area
      !that detects it: min and max in (X,Y,Z)
      call FindVDAreaThatDetects(curParticle)
      !Sweep the found 3D area
      do i= min_i, max_i
       do j= min_j, max_j
        do k= min_k, max_k
          call computeLOS-Distribution(i,j,k)
          call computeAbsorption-AtPixel(i,j,k)
          call computeEmission-AtPixel(i,j,k)
        enddo
       enddo
      enddo
    enddo
  enddo
enddo
do k=1, VD_TIME_RES
  do j=1, VD_VER_RES
    do i=1, VD_HOR_RES
     call solveRadiativeTransferEquationAtEachVDPixel
    enddo
  enddo
enddo
```

3. Parallel implementations and optimizations

A large number of scientific simulations, including radio-map computing algorithms, are array-based loop-intensive applications, formulated as an outer do loop that nests several parallel and serial inner do loops with different workloads and data access patterns. Usually, parallelism cannot be exploited at outer-loop level due to some specific sequential constraints. In this work, we evaluate the next three parallel models: data-sharing, data-privatizing and hybrid models for SPEV.

3.1. Data-sharing approach

This approach can be systematized by firstly, determining parallel inner do loops that can benefit from parallelism, i.e., loops that have a workload higher than the overhead involved by their parallel implementation [17], secondly, partitioning the iterations of the selected parallel loops among concurrent threads. Thirdly, privatizing auxiliary data that do not require global updates, i.e., reads and writes from multiple participating threads and, finally, protecting global shared data updates using locks. The most

computationally expensive loops in SPEV are evolve and emit do loops presenting around 25% and 65% respectively of the global cost in the serial implementation. After a careful analysis of both loops, these are some possible optimizations.

3.1.1. Optimization of evolve

The dense multi-dimensional matrices that accumulate the evolution of the particles and their corresponding emissions detected along the simulation are shared between threads. In each evolve phase, about 0.02% of the evolved particles go out of the considered domain and consequently their corresponding data are eliminated from the physical system. This operation causes accesses to main memory and therefore penalizes data locality. To generate an optimized sharing implementation of this stage, we reformulate *evolve* into two steps, a fully parallel *evolve* phase and a purely serial *compact* stage that is applied to each *n*th outer iteration (*n* is selected experimentally), as sketched below.

```
do(curParticule=1, NumParticles)
  call computeTrajectory(curParticle)
  if(curParticle is in the fluid) then
    call evolveEnergy(curParticle)
  else
    Id(curParticle)=0
    Density(curParticle)=0
  endif
enddo
!Each specific number of curFile call Compact()
if(counter .eq. n) then
  call Compact()
endif
```

3.1.2. Optimal loop mapping and critical section

In the body of the *emit* do loop, a number of successive accesses to the shared three five-dimensional matrices (i.e., the line-of-sight distribution matrix, the absorption coefficient matrix and, the emission coefficient matrix) have to be protected from simultaneous writes in the same memory position. The first and second dimensions of these matrices correspond to the number of the considered frequencies and virtual detector resolution which are equal to 3 and 3 respectively, while the last three dimensions represent the spatial VD area which is considered to be equal to $12 \times 36 \times 32$. The accesses to update the pixels of the shared arrays do not follow any order. Although, this critical section represents only 3% of the serial emit stage runtime, it becomes a clear bottleneck to the parallel implementation. We implemented two levels of critical section. The first uses one a single lock to protect all the arrays as sketched below.

```
acq =1
dowhile(acq .eq. 1)
   test&set(acq,lock)
enddo
do i
   do j
     do k
     !Compute distribution, absorption & emission
     call update(Array1(1:Freq,1:Res,i,j,k),...)
     enddo
   enddo
enddo
lock = 0
```

where *Freq* is the dimension of the frequency space and *Res* is the resolution of the VD. The second uses finer grain locks, where multiple threads can modify concurrently the same array but in different positions.

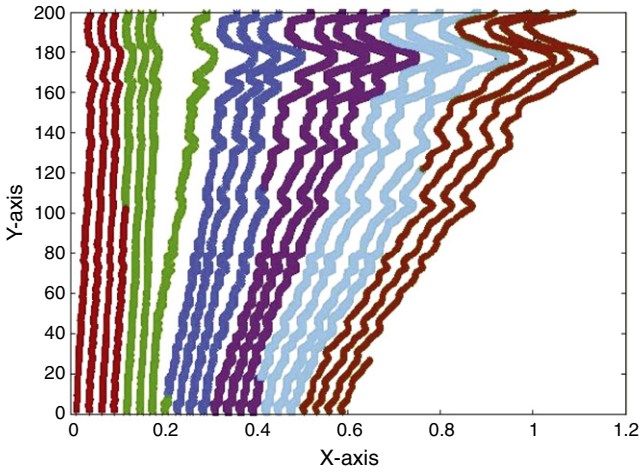

Fig. 3. An example of the resulting spatial distribution of the particles when considering six workers. Each color shows the particles of a specific process. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

```
acq =1
dowhile(acq .eq. 1)
   test&set(acq,lock(i,j,k))
enddo
 call update(Array1(1:Freq,1:Res,i,j,k),...)
 lock(i,j,k)=0
```

The iteration space of the parallel *evolve* loop and specially the *emit* do loop must be carefully partitioned and scheduled among the available cores. This process can be automated by running on each different architecture a number of executions that correspond to different mapping strategies and use different chunk sizes and at the end of the executions compare the runtimes and select the mapping strategy that gives the best performance. In next section, we only report the results of the scheduling strategy and chunk size that give the best results. Finally, the *Inject* phase introduces particles periodically and is carried out by only one thread due to its low workload.

3.2. Data-privatizing approach

At the initialization step, N_{spec} families of particles are created and introduced into the grid that simulates the hydrodynamic fluid. Afterwards, each single family generates a number of particles so that they fill their corresponding space delimited by $[x_i\ x_j]$ and $[y_i\ y_j]$, where i and j=1, N_{spec}. Recall that the number of replicas in 3-D of a particle increases with r, where r is the distance between a given particle and the z-axis. Therefore, to obtain a similar total number of particles per worker, and consequently a good load balance, we partition the N_{spec} families among all participating workers using a round robin assignment as summarized in the next pseudo-code.

```
subroutine create(N_spec)
  do i=Worker_id+1, N_spec, Num_Workers
    call createParticulesForFamilly(i)
  enddo
end
```

For illustration, an example of the resulting spatial distribution among six workers is shown in Fig. 3. Notice that the x-coordinates of all the particles are very close. That is, the total number of replicas (around the (0,0)) generated per worker is similar. In this approach, each worker allocates its own local data to record the evolution of its particles. Arrays that need global updates by

Table 1
Architectural summary of the three multi-core platforms used in the experiments.

Platform	#Sockets	#Cores/socket	SMT	L1D cache (kB)	L2 cache	L3 cache (MB)	Memory (GB)
Intel X5355	2	4	No	4 × 32	2 × 4 MB	–	16
Intel X7550	4	8	Yes	8 × 32	8 × 256 kB	1 × 18	128
AMD 6172	4	12	No	12 × 64	12 × 512 kB	2 × 6	128

all threads are replicated. Consequently, each process evolves its own set of particles, periodically injects its corresponding fraction of new particles and stores the computed line-of-sight, emission and, absorption coefficients in its local matrices. Once the iterative process is finished, an all-reduce operation is applied to obtain all the contributions of all the particles and finally the radiative transfer equation is solved at each VD pixel.

3.3. Hybrid approach

This approach applies the privatizing and sharing models at two levels successively. From a data point of view, this consists in applying a two-level decomposition strategy across the whole system. First, it performs a block decomposition among processes. Then, each block is partitioned in a round robin fashion among threads. This approach is more flexible than the aforementioned models in the sense that it allows adjusting/customizing the degree of data-sharing depending on the memory hierarchy characteristics of each architecture. The optimal mapping of the two-level decomposition among processes and threads will allow minimization of the negative NUMA effect and increasing the positive sharing effect between threads. The size of the blocks depends on the number of processes and threads used.

In other words, each process is assigned a set of particles of the same color as shown in Fig. 3 and then each sub-set of particles is assigned to the threads.

4. Experimental testbed

The experiments shown in this paper were carried out on three leading multi-socket, multi-core systems: (i) a dual-socket quad-core Intel X5355 (Clovertown), (ii) a quad-socket eight-core Intel X7550 (Beckton) and, (iii) a quad-socket 12-core AMD Opteron 6172 (Magny-Cours). An architectural overview and characteristics of each one of these systems are shown in Table 1 and Fig. 4.

4.1. Intel dual-socket quad-core Clovertown

The Intel Xeon 5300-series (Clovertown) is a quad-core successor of the dual-core 5100-series (Woodcrest) segment. It consists of two of those dual-core chips integrated in one multi-chip package. The specific model used in our experiments is X5355, which runs at 2.66 GHz and performs around 36 GFLOPS double-precision in the LINPACK benchmark.

Each core of the processor includes a private 32 kB L1 data cache, and each chip (two cores) shares a 4 MB L2 cache. The complete processor has access to a 1333 MHz front side bus (FSB) that connects to the external main memory. The processor cores do not support hyper-threading technology (there is one hardware thread context per core).

The multi-socket multi-core system used in our experiments includes two sockets containing an X5355 processor, each one with an independent FSB to connect to the shared 16 GB main memory, as shown in Fig. 4(a).

4.2. Intel quad-socket eight-core Beckton

The Intel Xeon 6500/7500-series (Beckton) is a Nehalem-based processor with up to eight cores; it uses buffering to support up to 16 DDR3 DIMMS per socket (no FSB). The specific model we use in our experiments is X7550 that runs at 2.0 GHz, includes eight cores and performs around 70 GFLOPS double-precision in the LINPACK benchmark.

Each processor core includes a private 32 kB L1 data cache, a private 256 kB L2 cache and each socket has a shared multi-banked 18 MB L3 cache, where only 16 MB of L3 are shared between the eight cores, the reminding 2 MB are dedicated to the directory. The processor also has four QuickPath interfaces (QPI), with a speed of 6.4 GT/s, that can be used to interconnect up to eight sockets. The processor cores support a two-way hyper-threading technology (there are two hardware thread contexts per core).

The system used in our experiments includes four sockets each containing an X7550 processor. The sockets are fully interconnected using QPI links and share a total of 128 GB main memory. Fig. 4(b) shows a block diagram of the X7550 processor (on the left) and a block diagram of the complete system (on the right).

4.3. AMD quad-socket 12-core Opteron Magny-Cours

The AMD Opteron 6100-series (Magny-Cours) is a multi-chip module (MCM) composed of two Instanbul-type processor dies. Each die of the 12-core version integrates six x86-64 cores, four x16 HyperTransport3 (HT3) ports and two DDR3 memory channels. Due to pin limitations, the MCM interfaces to four DDR3 channels and only four x16 HT3 ports (note that each x16 HT3 port can operate as two independent x8 HT3 ports). The specific model we use in our experiments is 6172 that runs at 2.10 GHz and performs around 182 GFLOPS double-precision in the LINPACK benchmark.

Each processor core includes a private 64 kB L1 data cache that is backed by a private 512 kB L2 cache (victim cache). The core also includes a shared 6 MB L3 cache, where 1 MB is reserved for a cache directory (HT Assist) for cache coherency purposes. Each core supports only one hardware thread context.

The system used in our experiments includes four sockets each containing a 6172 MCM. The sockets are strongly interconnected using x8 coherent HT3 links (on-package pair x16 and x8) and dedicate one x16 non-coherent HT3 link per socket for I/O. The sockets share a total of 128 GB main memory. Fig. 4(c) shows a block diagram of the 6172 processor die, the processor MCM and the complete system.

5. Results and analysis

The performance evaluations discussed in this section were carried out using the following configuration. For thread creation and management, we used OpenMP pragmas [18] and directives included in the Intel Fortran compiler [19]. In the sharing implementation, threads are created and joined once each outer iteration (i.e., loop with index *curFile*). This produces less overheads than creating and joining threads once during all the simulation due to the fact that the overheads due to synchronization in order to read radio-maps from disk are higher. Whereas, in the privatizing implementation, the processes are created using the standard MPI [20]. Thus, each process reads individually the input files. We used Intel ifort 10.1 with maximum optimization compilation option −O3.

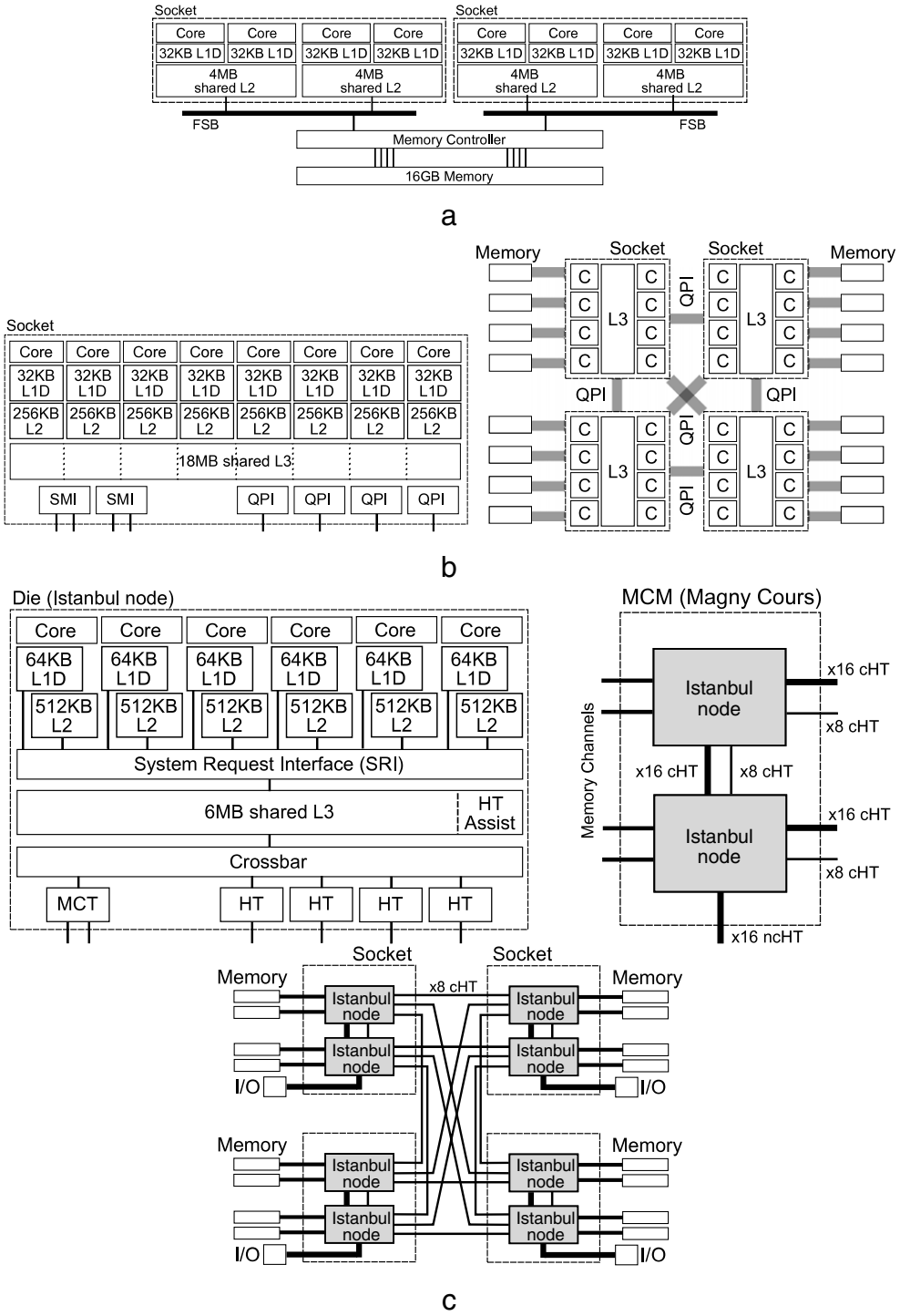

Fig. 4. Block diagrams of (a) dual-socket quad-core Intel X5355 (Clovertown), (b) quad-socket eight-core Intel X7550 (Beckton) and (c) quad-socket 12-core AMD Opteron 6172 (Magny-Cours).

The workload of each iteration *curFile* depends mainly on the maximum number of particles allowed to be created. The data read from radio-maps does not affect the load. Thus, to accelerate the performance evaluation process, we used a typical input size (four families of four million particles) but compute the emission using a small set of radio-maps, two randomly selected radio-maps, from a total number of 25 radio-maps. The performance results do not include the radiative transfer equation solution, since it is solved only once, at the end of the simulation and, because its load is negligible compared to the main loop load.

All the executions were conducted using the taskset Linux command to limit the executions on the considered set of cores and avoid thread migration to cores that are not included in that set. To increase data-sharing between threads that share a certain cache level, we ensure that these threads run on the same cores throughout the execution by binding them to cores using the Pthreads function, pthread_setaffinity_np() [21]. The number of cycles per core and L3 misses presented in this section were measured by counting PAPI_TOT_CYC and the native LLC_MISSES events using the PAPI portable interface to hardware performance counters [22].

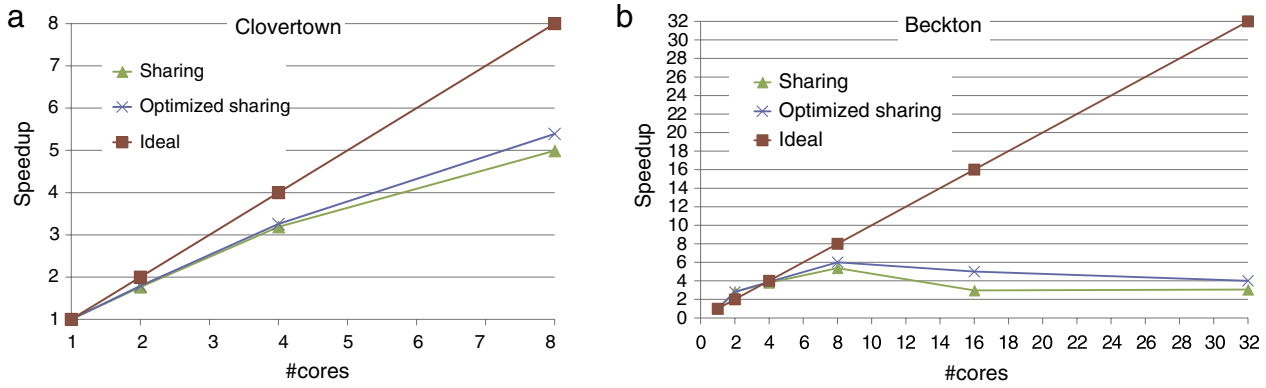

Fig. 5. Speedup of the sharing (# threads = # cores) and optimized sharing (# threads = # cores) implementations on dual-socket quad-core Clovertown (a) and Intel quad-socket eight-core Beckton.

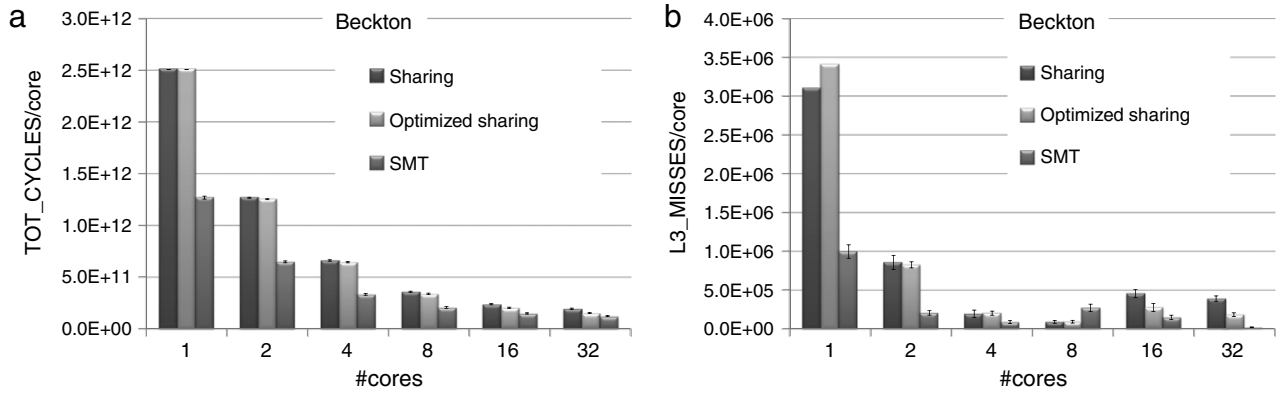

Fig. 6. The number of cycles per core (a) and LLC-misses (b) of the sharing (running one thread/core), optimized sharing (running one thread/core) and, optimized sharing implementation that exploits SMT support (running two threads/core) on Intel quad-socket eight-core Beckton. The error bars represent the absolute difference between the maximum and the minimum number of cycles and LLC-misses respectively among all the considered threads.

5.1. Sharing versus optimized sharing and SMT impact

The results presented in this section were obtained using the parameters that experimentally give the best performance. That is, by applying the serial *compact* stage only once each 10 *curFile* iterations and using the optimal scheduling for *evolve* and *emit* parallel do loops reduces substantially the overhead due to waits. In fact, preliminary runtime results show that, by using a round robin scheduling with a chunk size equal to one together with binding threads to work on the same core during all the execution gives the best performance. This can be explained by the fact that this configuration maximizes sharing opportunities at the LLC and because the slight load-unbalance due to static scheduling allows a better overlapping between computation inside and outside the critical section in the emit do loop, and therefore, minimizes the waits at the critical section. In fact, using a dynamic scheduling to balance the load worsens the scalability of the emit do loop earlier.

An evaluation of the scalability of the sharing implementation, based on coarse grain locks, and its optimized version, based on fine grain locks, on the dual-socket quad-core Clovertown, and Intel quad-socket eight-core Beckton are shown in Fig. 5(a) and (b) respectively. Moreover, an evaluation of the number of cycles and LLC-misses of the sharing, optimized sharing and, optimized sharing implementation when exploiting the Simultaneous Multi-threading (SMT) support (by running two threads per core) on Intel quad-socket eight-core Beckton, is shown in Fig. 6(a) and (b) respectively. Recall that Clovertown and Magny-Cours architectures do not support SMT technology. To quantify load unbalance between threads in each run, Fig. 6(a) and (b) show in addition to the number of cycles and L3-misses, the differences between the maximum and the minimum number of cycles and LLC misses per execution respectively (drawn as error bars). Notice that these differences are insignificant, which means that the three versions are well balanced.

As it can be seen from Fig. 5(a) and (b), sharing and optimized sharing implementations show a good and similar scalability at one socket level, since both, the number of cycles and L3-misses are similar too. However, when the evaluation includes cores from different sockets, the scalability of both implementations starts to decay substantially on both platforms, especially the sharing version. This is due to the fact that the bandwidth of the connections between sockets is lower than the bandwidth inside the socket and also to the increment in L3-misses when the number of cores > 8, in the case of the Beckton based platform.

Nevertheless, the optimized sharing version based on finer locks scales better, but not well enough, with respect to the coarse grain based version, reaching an improvement of 8% on eight cores in the Clovertown based platform and 20% on 32 cores in the Beckton based platform. This slight improvement can be explained not only by the decrease in wait times to acquire the locks but also by a slightly better L3 cache management (as shown in Fig. 6(b)). The difference in LLC-misses between the sharing and the optimized sharing versions on a single core is due to the use of additional memory for the two-dimensional matrix of locks in the optimized version.

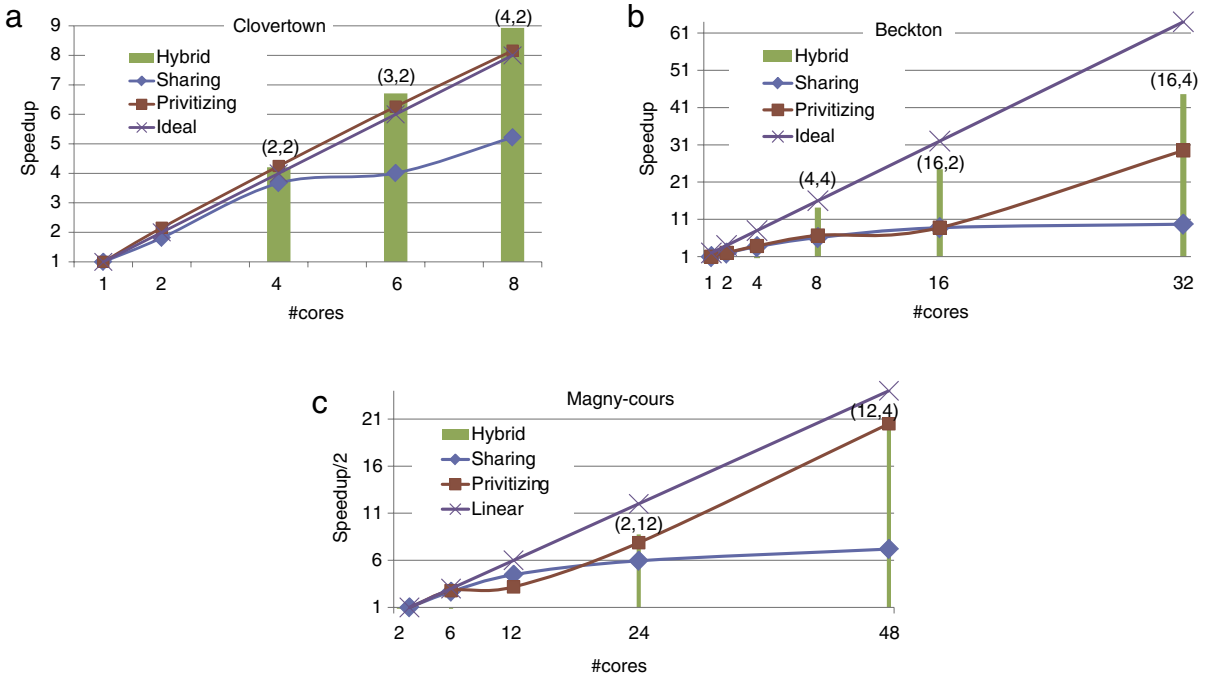

Fig. 7. Speedup comparison between optimized sharing, privatizing and hybrid versions on (a) dual-socket quad-core Clovertown, (b) Intel quad-socket eight-core Beckton and, (c) quad-socket 12-core Magny-Cours.

Exploiting SMT by running two threads per core on the Beckton based system shows a speedup up to two times larger than the speedup of the optimized sharing implementation on one socket. This improvement at the one socket level is due mainly to the increase of data-sharing in L1, L2 and L3 levels, specially between threads that run on the same core, which implies a decrease of the LLC-misses as can be clearly seen in Fig. 6(b). However, although exploiting SMT improves the scalability by 27% (with respect to the optimized sharing version) on four sockets, the speedup is still poor (i.e., speedup on 32 cores = 17). In the next subsection, we compare the performance of the optimized sharing implementation versus the privatizing and hybrid approaches. For the Beckton based system, we consider the version that exploits SMT.

5.2. Comparison between sharing, privatizing and, hybrid approaches

The scalability results of the optimized sharing, privatizing and hybrid implementations, on the dual-socket quad-core Clovertown, quad-socket eight-core Beckton and, quad-socket 12-core Magny-Cours, are shown in Fig. 7(a), (b) and, (c) respectively. For the hybrid implementation, we plotted only the best speedups obtained on the number of cores indicated in the x-axis. These speedups correspond to a specific combination of processes and threads, indicated as (p processes, t threads) in Fig. 7. On the Magny-Cours based platform, the serial version was excessively slow due to memory limitations. For this reason, in this case, we calculated the speedup with respect to the runtime on two cores.

In general, the hybrid strategy clearly outperforms both sharing and privatizing approaches on all the used systems especially on the Beckton based platform improving the speedup by a factor of 66% (on 32 cores) over the next best implementation, i.e., the privatizing version. In particular, on the systems that do not support SMT, the three implementations show competitive scalability at one socket level, i.e. on a single Clovertown and a single Magny-Cours. More similar scalability is obtained when the participating cores have a shared LLC, i.e., for core numbers equal to 2 and 6 in Clovertown and Magny-Cours respectively. However, when the evaluation extends to cores of different sockets, the scalability of the sharing version drops drastically especially on the Magny-Cours based system; while the privatizing and hybrid implementation maintain a good speedup over all the sockets reaching a speedup$_2$ of about 20 on 48 cores (the ideal speedup in this case is equal to 24).

The hybrid implementation with the execution configuration that maps four processes, each process creates two threads (four processes, two threads), on eight cores, on the Clovertown based system, reaches a speedup 9% better than the privatizing one. The high scalability of the privatizing strategy is mainly due to the fact that communications are required only at the end of the simulation. The hybrid implementation inherits the same characteristic from the privatizing one with the additional advantage of minimizing LLC interferences between threads that share data and work.

On the Beckton based system, the hybrid implementation noticeably outperforms the privatizing and sharing ones reaching speedups larger than the number of cores used. In contrast to the sharing and privatizing model, the hybrid model is the only approach that exploits better SMT technology and consequently improves drastically the scalability. Moreover, in all the considered systems, on each determined number of cores, there exists a specific combination (p processes, t threads) that performs the best. For example, in used dual-socket quad-core Clovertown, with Uniform Memory Access, the combination (p processes, two threads) scales the best on $p \times 2$ cores. The number of processes p that should be created can be predictable. While on used Intel quad-socket eight-core Beckton and AMD quad-socket 12-core Opteron Magny-Cours, with Non Uniform Memory Access, the optimal configuration (p processes, t threads) is variable but can be found experimentally via auto-tuning by searching among a reduced search-space, where t takes values from 2 to the total number of cores per socket ($\times 2$ in the case of SMT).

5.3. Numerical results

Shaping the spectral evolution simulations needs performance of many executions of the parallel hybrid spectral evolution

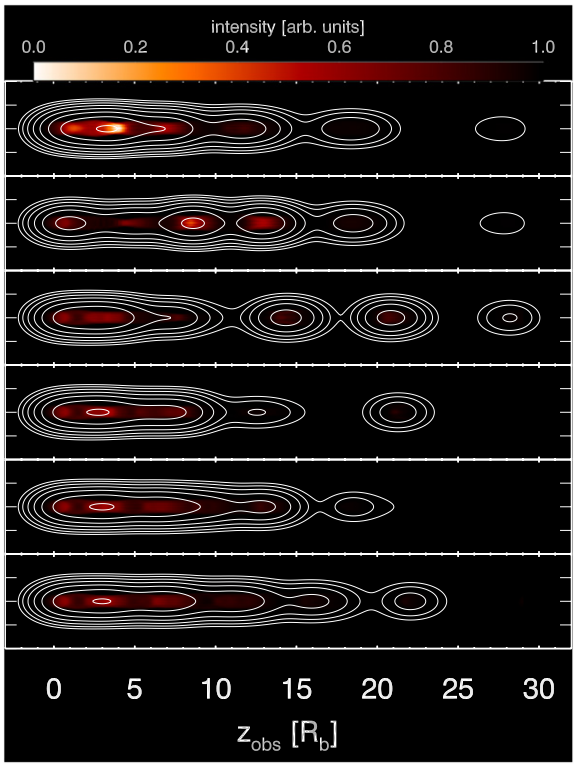

Fig. 8. Snapshots of the emission at 43 GHz due to component evolution computed by hybrid implementation of the SPEV method. Z_{obs} is the distance from the jet core projected to the observer's plane of the sky.

code, on the three multi-socket multi-core systems, with different physical corrections, till the results are as similar as possible to the real VLBI-maps. An illustration of a VLBI-map calculated by our hybrid implementation is shown in Fig. 8.

6. Conclusions and future works

In summary, for VLBI-resolution radio-map algorithms and other algorithms with similar structure, the data-sharing programming model represents a good solution but only in small scale systems that do not support SMT. In addition, to obtain competitive scalability with the privatizing one at one socket level, several optimizations must be applied.

To substantially facilitate tuning the sharing implementation, two aspects can be automated:

- The process of searching for parallel loops that can benefit from parallel computing can be implemented by checking that their workload is larger than the overhead involved by their parallel implementation. In general, thread creation and mapping incur an overhead on the order of 2 k cycles [17] (this quantity depends on the architecture of the processor and the used software). Further, this overhead increases when the number of threads increases.
- The iterative searching for the best combination scheduling-chunk size can also be automated using tuning methods based on profiling.

The data-privatizing programming model is able to maintain reasonable scalability on medium to large scale multi-core systems because it preserves parallelism along the simulation and minimizes the communication overheads. Nevertheless, this approach is unable to benefit from exploiting SMT.

Finally, the hybrid model is the most appropriate alternative on multi-core multi-socket systems since it allows (i) finding an optimal configuration that optimizes the LLC management by minimizing interferences between threads in the same socket and (ii) preserving parallelism. The selection of the best mapping can also be automated by running the parallel hybrid implementation using a determined set of c possible combinations, where $c = 2$ to # allowed threads at one socket level. Furthermore, this strategy can be easily ported to heterogeneous platforms.

Future works will explore and implement auto-tuning heuristics that find the best combinations (# processes, # threads) on a wider range of multi-core multi-socket platforms. In addition, in collaboration with the Relativistic Astrophysics and Cosmology Research Group of the University of Valencia, we are currently, working on implementing the polarized radiation transfer equation in the hybrid implementation of SPEV by giving the magnetic field in the fluid a more ordered structure. We expect to obtain a polarized synchrotron emission instead of the non-polarized one that we have been computing so far. We expect the computation of the emission and absorption coefficients as well as solving the radiative transfer equations to be more computationally expensive than in the current simulation. The results reported in this work will be the basis of our future parallel simulation. The inclusion of the polarized radiation transfer is an important next step since the knowledge of the degree of the polarization can provide important clues about the orientation and structure of the (unobservable) magnetic field in relativistic jets. Being able to determine in more detail the magnetic field of relativistic jets provides important information about the process which generates them, i.e. about the central regions of the AGNs.

Acknowledgments

This work was supported by the Spanish Ministry of Education and Science through the Juan de la Cierva contract, grant CSD2007-00050 and, the project TIN2006-01078 and, through the European Research Council grant CAMAP-259276.

References

[1] M. Kandemir, S.P. Muralidhara, S.H.K. Narayanan, Y. Zhang, O. Ozturk, Optimizing shared cache behavior of chip multiprocessors, in: 42nd International Symposium on Microarchitecture, MICRO'09, New York, NY, December 2009, pp. 505–516.
[2] A. Jaleel, M. Mattina, B. Jacob, Last Level Cache (LLC) performance of data mining workloads on a CMP—a case study of parallel bioinformatics workloads, in: 12th Int'l. Symp. on High Performance Computer Architecture, HPCA'06, Austin, TX, February 2006, pp. 88–98.
[3] D. Tam, R. Azimi, L. Soares, M. Stumm, Managing shared L2 caches on multicore systems in software, in: Workshop on the Interaction between Operating Systems and Computer Architecture, WIOSCA 2007, in Conjunction with ISCA'07, San Diego, CA, June 2007.
[4] P. Mimica, M.A. Aloy, I. Agudo, J.M. Martí, J.L. Gomez, J.A. Miralles, Spectral evolution of superluminal components in parsec scale jets, The Astrophysical Journal (696) (2009) 1142–1160.
[5] J.L. Gómez, J.M. Martí, A.P. Marsher, J.M. Ibáñez, J.M. Marcaide, Parsec-scale synchrotron emission from hydrodynamic relativistic jets in active galactic nuclei, The Astrophysical Journal Letters (1995) 19–21.
[6] J.L. Gómez, J.L. Martí, J.M. Marsher, A.P. Ibáñez, J.M. Marcaide, Hydrodynamical models of superluminal sources, The Astrophysical Journal (449) (1997) 33–36.
[7] T.W. Jones, D. Ryu, A. Engel, Simulating electron transport and synchrotron emission in radio galaxies: shock acceleration and synchrotron ageing in axisymmetric flows, The Astrophysical Journal 512 (1) (1999) 105–124.
[8] I.L. Tregillis, T.W. Jones, D. Ryu, Simulating electron transport and synchrotron emission in radio galaxies: shock acceleration and synchrotron ageing in three-dimensional flows, The Astrophysical Journal 557 (2001) 475.
[9] I. Agudo, J.L. Gomez, J.M. Marti, J.M. Ibanez, A.P. Marscher, A. Alberdi, M.A. Aloy, P.E. Hardee, Jet stability and the generation of superluminal and stationary components, The Astrophysical Journal (2001) 183–185.
[10] F. Casse, A. Marcowith, Relativistic particle transport in extragalactic jets–I. Coupling MHD and Kinetic theory, Astronomy and Astrophysics 404 (2003) 405–421.
[11] A. Ferrari, Modeling extragalactic jets, Annual Review of Astronomy and Astrophysics (36) (1998) 539–598.

[12] M. Boettcher, D.E. Harris, H. Krawczynski (Eds.), Relativistic Jets from Active Galactic Nuclei, Wiley, 2012.
[13] A.C.S. Readhead, P.N. Wilkinson, The mapping of compact radio sources from VLBI data, The Astrophysical Journal 223 (Part 1) (1978) 25–36.
[14] J.A. Miralles, K.A. van Riper, J.M. Lattimer, The Boltzmann equation in general relativistic rotating systems: cooling of rotating neutron stars, The Astrophysical Journal 407 (1993) 687.
[15] G.M. Web, Relativistic transport theory for cosmic rays, The Astrophysical Journal 296 (1985) 319.
[16] J.G. Kirk, in: A.O. Benz, T.J.-L. Courvoisier (Eds.), Plasma Astrophysics: Saa-Fee Advanced Course, Springer-Verlag, 1994, p. 225.
[17] A. Kejariwal, A. Nicolau, A.V. Veidenbaum, U. Banerjee, C.D. Polychronopoulos, Efficient scheduling of nested parallel loops on multi-core systems, in: 38th Int'l. Conf. on Parallel Processing, ICPP'09, Vienna, Austria, September 2009, pp. 74–83.
[18] http://openmp.org/wp/openmp-specifications/.
[19] Intel compilers for Linux. http://www.intel.com/cd/software/products/asmo-na/eng/compilers/284264.htm.
[20] http://www.mcs.anl.gov/research/projects/mpi/.
[21] D.R. Butenhof, Programming with POSIX Threads, 1997.
[22] Performance application programming interface. Available in http://icl.cs.utk.edu/papi/.