Accelerating analysis of void space in porous materials on multicore and GPU platforms

Richard L. Martin[1], Prabhat[1], David D. Donofrio[1], James A. Sethian[1,2], and Maciej Haranczyk[1]

[1] Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA

[2] Department of Mathematics, University of California, Berkeley, CA 94720, USA

Acknowledgment We would also like to thank NERSC for access to GPU computing facilities. This work was supported by the Director, Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231 through the Scientific Discovery through Advanced Computing (SciDAC) program's Visualization and Analytics Center for Enabling Technologies (VACET). This work was supported in part by the Applied Mathematical Science subprogram of the Office of Energy Research, U.S. Department of Energy, under Contract Number DE-AC03-76SF00098, and by the Computational Mathematics Program of the National Science Foundation.

Running head
Analysis of porous materials

Richard L. Martin
Computational Research Division
Lawrence Berkeley National Laboratory
One Cyclotron Rd, MS 50F-1650
Berkeley, CA 94720, USA
E-mail: richardluismartin@lbl.gov
Ph. +1 (510) 486-7749

Prabhat
Computational Research Division
Lawrence Berkeley National Laboratory
One Cyclotron Rd, MS 50F-1650
Berkeley, CA 94720, USA
E-mail: prabhat@hpcrd.lbl.gov
Ph. +1 (510) 486-7752

David D. Donofrio
Computational Research Division
Lawrence Berkeley National Laboratory
One Cyclotron Rd, MS 50F-1650
Berkeley, CA 94720, USA
E-mail: ddonofrio@lbl.gov
Ph. +1 (510) 486-5518

James A. Sethian
Computational Research Division
Lawrence Berkeley National Laboratory
One Cyclotron Rd, MS 50A-1148
Berkeley, CA 94720, USA

and

Department of Mathematics
University of California, Berkeley
Berkeley, CA 94720, USA

E-mail: sethian@math.berkeley.edu
Ph. +1 (510) 486-6006

Maciej Haranczyk (Corresponding Author)
Computational Research Division
Lawrence Berkeley National Laboratory
One Cyclotron Rd, MS 50F-1650
Berkeley, CA 94720, USA
E-mail: mharanczyk@lbl.gov
Ph. +1 (510) 486-7749

Abstract. Developing computational tools that enable discovery of new materials for energy-related applications is a challenge. Crystalline porous materials are a promising class of materials that can be used for oil refinement, hydrogen or methane storage as well as carbon dioxide capture. Selecting optimal materials for these important applications requires analysis and screening of millions of potential candidates. Recently, we proposed an automatic approach based on the Fast Marching Method (FMM) for performing analysis of void space inside materials, a critical step preceding expensive molecular dynamics simulations. This breakthrough enables unsupervised, high-throughput characterization of large material databases. The algorithm has three steps: (1) calculation of the cost-grid which represents the structure and encodes the occupiable positions within the void space; (2) using FMM to segment out patches of the void space in the grid of (1), and find how they are connected to form either periodic channels or inaccessible pockets; and (3) generating blocking spheres that encapsulate the discovered inaccessible pockets and are used in proceeding molecular simulations. In this work, we expand upon our original approach through (A) replacement of the FMM-based approach with a more computationally efficient flood fill algorithm; and (B) parallelization of all steps in the algorithm, including a GPU implementation of the most computationally expensive step, the cost-grid generation. We report the acceleration in each step and in the complete application achievable, and discuss the implications for high-throughput material screening.

Keywords: Porous Materials, Zeolites, Screening, General Purpose Computation on GPUs, Multicore Programming, Code Optimization

1 Introduction

Crystalline porous materials have found wide use in industry since the late 1950s. They are commonly used as chemical catalysts, in particular as cracking catalysts in oil refinement, membranes for separations and as water softeners[1–4]. There is an increasing interest in utilizing these materials as membranes or adsorbents for carbon dioxide capture applications as well as for hydrogen or natural gas storage[5–9]. The need for optimal materials, which would be inexpensive, safe and efficient, has stimulated researchers to develop databases containing millions of predicted material structures[10–12]. Development of such databases holds great promise for discovery of new materials for many applications. However, it is now being realized that in order to make such discoveries possible, new computational and cheminformatics techniques have to be developed to characterize, categorize, and screen such large databases. Approaches based on manually intensive visual analysis techniques are simply infeasible for screening large numbers of structures in a high-throughput manner.

An important aspect of analysis of porous materials is determination of a guest molecule-related accessibility of their void space. In particular, this involves the detection of inaccessible pockets, which can be occupied by guest molecules in computer calculations, even though such pockets are inaccessible in adsorption experiments. It is important to account for (and often artificially block or exclude) these pockets in the calculation of guest-accessible volumes or surface areas, or the prediction of guest-related properties using molecular simulation techniques. For example, in Monte-Carlo (MC) simulations of adsorption[13], the blocking procedure can be a simple distance-check from the center of the small pockets and a rejection of all Monte-Carlo trial moves that would place a molecule inside a certain radius[14]. The latter sets of positions of centers of pockets and the corresponding radii are referred to as the blocking spheres. The importance of pore blocking in Monte-Carlo simulations has been recently re-emphasized by Krishna and van Baten[15]. Similar to MC simulations, molecular dynamics simulations have to account for such pockets in order to ensure that guest molecules are not placed in inaccessible areas.

Recently, we presented an automatic approach for determining accessibility of the void space of porous materials, thereby enabling execution of molecular simulations-based calculation of materials' properties in an unsupervised, high-throughput manner[16]. In our approach, we used a partial differential equations (PDE)-based front propagation technique to segment out channels and inaccessible pockets of a periodic unit cell of a material. Unlike other approaches that approximate guest molecules with a spherical probe, the general framework of our approach[17] allows guest molecules to have more complex and flexible shapes. We model complex objects built from solid blocks connected by flexible links (molecular worms). The new

capabilities offered by our approach[16, 17] come at the price of increased computational cost associated with the discretization and analysis of the configuration space of a guest molecule inside the material.

We make three key observations regarding our approach which enable us to effectively utilize modern high-performance computing platforms. Firstly, our approach can be easily applied to different materials in the database in a distributed setting. Since analysis of one material is independent of other materials, each material can be processed on a single node with no inter-node communication requirements. Secondly, our approach is memory intensive; we typically require 4 to 64 GB of memory for processing a single material. This amount of memory is typically available on a single HPC node, but is usually shared across multiple CPU cores. Sometimes HPC nodes might also have a powerful co-processor card (GPU or vector processor) available with much less memory and limited bandwidth. We need to be cognizant of this trend in modern HPC systems and utilize all available computational horsepower on a single node. Finally, the most computationally expensive step in our analysis (i.e. cost grid calculation) has ample parallelism in its formulation: the step can proceed independently at each discretization of the configuration space.

Following these observations, we have refactored our algorithms and implemented optimized versions on an AMD multicore CPU and an Nvidia GPU. We discuss specific details of our algorithm and demonstrate dramatic speed-up over our serial baseline implementation in the following sections.

2 Related Work

Traditional approaches for analyzing accessibility of the void space in crystalline porous materials involve visual analysis. Such analyses are typically performed by inspection of the so-called pore landscapes[15], which are isosurfaces corresponding to the maximum accessible free energy level[18]. An alternative approach involves analysis of abstract structure representations such as chemical hieroglyphs[19]. Automatic detection of internal cavities has been explored in the context of zeolites[20] - using 'largest void cylinders' and spherical 'cages' - and proteins[21], and the more general question of finding possible pathways through chemical systems has been addressed in both proteins[22] and materials[23–26] . All of the aforementioned references, with the exception of [17], attempt to study paths of a spherical probe representing the molecule inside a convex hull constructed from atoms of a protein or materials framework.

To the best of our knowledge, all of the approaches for automatic analysis of porous materials are designed for single-threaded execution. We are not aware of a study which implements these techniques on heterogeneous computing architectures and discusses performance characteristics. We do note, however,

that the field of computational chemistry routinely uses HPC systems (including multicore CPUs and vector machines) for determining electronic structure and running molecular dynamics simulations. GPU platforms have been recently utilized in the latter applications[27–30], and in an evolutionary algorithm for zeolite framework generation [31–33].

3 Science Application

We have recently developed an approach to analyze accessibility of the void space in a porous material to a guest molecule. Our approach can accommodate simple approximations of a guest molecule as a spherical probe, as well as more complex models resembling the shape and flexibility of a "real" molecule[17]. In the latter model, complex objects are built from solid blocks connected by flexible links, and they are able to change orientation and/or shape during the traversal of a chemical structure, allowing them to reach areas not accessible to either a single large spherical probe or rigid real-shape probes.

In the above approach, we cast this problem as an Hamilton-Jacobi-type Eikonal equation in configuration space describing a guest molecule inside a material:

$$|\nabla U| = C(x).$$

Here, U is the minimal total cost and $C(x)$ is a cost function defined at each point x in the domain, corresponding to its ability to be occupied. The equation is solved by using a variant of Fast Marching Methods(FMM),[34, 35] which are Dijkstra[36]-like methods to solve the boundary value problems of the form of the Eikonal equation. Here, abstractly, the cost function is defined at the beginning of the problem, and the solution $U(x)$ to the above problem represents the total cost, which is the smallest obtainable integral of $C(x)$, considered over all possible trajectories throughout the computational domain from a start point to a finish point. The latter feature can then be used to construct practical techniques enabling, for example, the determination of shortest paths[17], and a prediction of the accessibility of sections of the void space, e.g. detection of inaccessible pockets[16].

The latter application of our approach, being of special interest in this article, has three steps:

1. Calculation of $C(x)$ cost at each grid point x of the discretized domain. Point x corresponds to a configuration of a probe inside a material, and in the simplest case of a spherical probe it corresponds to a position inside a material. The time requirement scales linearly with N, where N is the total number of mesh points in the computational domain.

2. Segmenting the grid into distinct, aperiodic regions of void space (patches). An illustration of this process is provided in Figure 1. After segmentation, periodic boundary conditions are applied, and regions which connect across the periodic boundary are connected. Thus, those regions which constitute channels through the void space can be identified, with the remaining regions constituting inaccessible pockets.
3. Calculation of blocking spheres to be used as 'exclusion zones' in the proceeding molecular simulations. Blocking spheres enclose all pockets of Step (2) completely, without intruding upon any channels of the system.

In this work we take the above application framework as a serial baseline prototype, and improve upon it in two major ways. Firstly, we replace the FMM-based method in the segmentation step with a non-PDE-based flood fill (also known as seed fill or bucket fill) algorithm. Flood fill is a technique whereby connected regions of adjacent grid points are efficiently identified, and is in common usage, for instance in image processing applications. This change constitutes a speed-up since the elements of the original algorithm associated with the total cost and casuality of propagating fronts are removed at the cost of not being able to predict shortest paths. Secondly, we develop and implement parallel algorithms for each of the three steps.

The absolute runtime for Steps (1)-(3) of the approach depends on factors including the number of grid points used to describe the system. The grid resolution depends on the requested accuracy and specific characteristics of the investigated material (e.g. density, which correlates with volume of the void space to be investigated), see Ref [17, 16, 26] for detailed discussion. We profiled typical "'production"' runs of our serial baseline implementation and discovered Step (1) to be the most time consuming, typically requiring an order of magnitude more time than Steps (2) or (3). This trend only gets worse as the size of the problem under investigation increases. Figure 2 illustrates this point with the example of four zeolite materials in which we detected void space pockets inaccessible to methane. These examples involved the calculation of a three-dimensional grid with step size of 0.1Å; due to the different sizes of the systems (size of the periodic box representing crystalline material) for EDI, HEU, TSC and LTN zeolite materials, this translates to approximately 300K, 2.5M, 29M and 45M grid points in each structure respectively. It can be seen that as the volume of the structures increases, the proportion of time spent on Steps (1) and (3) increases, while the proportion for Step (2) decreases.

Within our aim of parallelizing the whole algorithm, our strategy in this work was to primarily focus on optimizing the most time-consuming task, i.e. calculating the cost function on the discretized configuration grid, which is also the most intrinsically parallel step. In the following, we will brief the calculations involved in each step. Further details can be found in [17].

The most straightforward cost function $C(x)$, where x is a grid point, is defined as follows: $C = 1$ for each point x which can be occupied, $C = \infty$ otherwise. Here, an "occupiable" point means a position in which the probe is not colliding with any atom of the material's framework; a collision occurs when the distance between the center of the probe and any of the structure atoms is smaller than the sum of their specified radii. The latter case is equivalent to a hard sphere approximation of atoms. More advanced definitions of the cost functions (including considerations of specific interactions) have also been proposed[17, 16].

Pseudocode for the cost-grid algorithm is given below. The algorithm uses two sets of coordinates: fractional and Cartesian. Fractional coordinates are used to determine the position of an atom within a periodic unit cell describing a crystalline porous material. These coordinates can easily handle periodic boundary conditions. Cartesian coordinates are used to determine distance between atoms.

```
For each discrete grid point
   If any of its fractional a,b,c coords not in range 0-1
      Output distance of zero (i.e. position is out of bounds)
   Else
      For each atom in probe molecule
         For each atom in structure
            Get fractional a,b,c difference between probe and structure atom coords
            Periodicise the difference
            Calculate Cartesian x,y,z distance
            Subtract atom radii
            Update minimum distance so far
            If minimum distance < zero
               Output distance of zero
         (end structure loop)
      (end probe loop)
      Output minimum distance
   (end if/else)
```

```
(end)
```

The nested loops in the above pseudocode, and the independence of each grid point calculation, make this algorithm amenable for parallelization; furthermore, the requirement to calculate $C(x)$ for up to millions of grid points, makes this algorithm also an interesting candidate for a GPU implementation. We give details of these implementations in the following section. The two remaining steps in the application are also parallelizable, and we provide pseudocode for the segmentation step below. The sphere blocking algorithm for inaccessible pockets has previously been described in detail[16]. The parallel implementation of steps (2)-(3) will be discussed in sections 4.7-4.8.

The segmentation steps identifies a set of distinct segments (patches) of the void space and assigns them unique identifiers (SEGMENTNUMBERs). The flood fill step requires keeping track of a "set" value for each grid point, which during the fill denotes whether this point has been visited (KNOWN), considered (TRIAL) or otherwise (FAR). Initially, all points with $C(x) > 0$ are FAR, as they are occupiable but the segment to which they belong is unknown. When a point is visited, it becomes KNOWN; TRIAL points are those that have been found to be adjacent to some KNOWN point, and are waiting in a stack to have their adjacent points considered in turn. After a segment has been filled, all points therein are assigned to KNOWN+SEGMENTNUMBER for future identification of to which segment (i.e. set-KNOWN) they belong.

```
Set FAR 0

Set TRIAL 1

Set KNOWN 2

Set SEGMENTNUMBER 1

For each discrete grid point, x

   If x is FAR

      Assign x to be TRIAL

      Push x to stack

      While stack is non-empty

         Pop the stack, retrieving point y

         Assign y to be KNOWN
```

```
         For each point z adjacent to y
            If z is FAR
               Assign z to be TRIAL
               Push z to stack
            (end if)
         (end adjacent points to stack loop)
      (end stack pop loop)
      Now stack exhausted, assign all KNOWN points to KNOWN+SEGMENTNUMBER
      Increment SEGMENTNUMBER
   (end if statement for this segment)
(end loop over entire grid)
```

Following the segmentation, segments which are adjacent through consideration of the periodic boundary are detected, and merged. At this point, segments can be classified as either channels or pockets. If it is possible to travel within one segment from one unit cell to the same position in an adjacent cell, then this segment is a channel; this property is determined through consideration of the periodic boundary adjacency matrix which encodes the connections between segments and which boundaries are crossed. All non-channel segments must be pockets, and will be blocked; note that it is possible for a series of segments to be connected across the periodic boundary and yet not form a channel (i.e., they form a periodic pocket).

4 Methods

We implemented optimized versions of the steps detailed in the previous section and tested performance on a high-end workstation with Opteron Magny-Cours processors. We also tested performance of the code on an Nvidia Fermi C2050 card. We chose these particular hardware platforms because they are representative of single nodes in our larger HPC facilities. Therefore, we can leverage the results from our present work for future large concurrency runs.

4.1 Dataset

For testing purposes, we chose the most challenging structure with the largest periodic unit cell in the IZA dataset of zeolites, LTN. Its unit cell is orthogonal and cubic, comprising 2304 atoms, of which 1536 (two thirds) are oxygen, and 768 (one third) are silicon. We sample the periodic unit cell at a 0.1Åresolution, resulting in a grid of 357 elements in each dimension (45,499,293 grid points total). The probe used to examine this structure is Methane, represented as a single sphere of radius 1.825Å(united atom approximation).

For each grid point, in addition to the minimum distance as described in the psuedocode, the algorithm must output a "set" (i.e. accessible, inaccessible or out of bounds). These two large arrays of values are encoded as character and unsigned short int variables respectively, resulting in a total of ∼136.5 million bytes (130.2MB) of data for this problem size. In the case of the multicore CPU code, this data is written directly into main memory and is analyzed by subsequent steps in our program (i.e. segmentation, etc). However, for the GPU implementation, partial results for the grid are computed, and then read back and placed into main memory on the host.

4.2 Software

We developed our code in C++ and used pthreads for the multi-threaded implementation. We used the gcc compiler with -O3 options for compiling optimized code. We also performed manual loop unrolling and use SSE intrinsics for using SIMD hardware on the CPU. We used CUDA for porting our code to the GPUs. Specific GPU optimizations are listed in a following section. Our code uses single precision floating point on both CPU and GPU platforms. We verified the results of our optimized CPU and GPU implementations against a gold-standard unoptimized CPU code, and checked that our results were consistent.

4.3 Hardware

We were primarily interested in optimizing the expensive cost calculation step, i.e. Step (1), on a multicore CPU and GPUs. Details pertaining to the hardware architecture for the Magny-Cours, FX5800 and Fermi C2050 are presented in Table 1. Extensive descriptions of GPU hardware and programming optimizations are available in [37–39].

4.4 Cost Calculation Step - Algorithm Characteristics

We profiled the cost-calculation step in the program and determined that it was compute-bound. The inner-most loop of the algorithm presented in Section 3 is the most expensive, involving calculation of distances

over thousands of structure atoms. We are able to fit data corresponding to atom positions in the L1 cache on the CPU, and constant memory on the GPU. We are primarily limited by the performance of the floating point units.

The size of the computational grid can be large (∼16GB), and this typically does not fit on GPU memory. We overcome this limitation by double buffering: we compute different parts of the grid while the results from the first part are read back asynchronously. The proportion of time spent in transferring data over the PCIe bus to the host memory is small compared to the cost of computing the grid; thereby we avoid both major limitations that plague typical GPGPU applications.

4.5 Optimizations for multicore CPU implementation

For the multi-threaded CPU implementation, we create a pool of threads (with the number of threads being equal to the number of cores). We direct each thread to analyze slices of the sampling grid where x is constant (i.e. x-slices). The first thread is initialized with $x = 0$, and each subsequent thread is initialized with the next unclaimed value for x. Each thread then loops over the other dimensions of the grid, beginning with the slowest (the y dimension) and ending with the probe orientation or conformation dimensions, if necessitated by the probe molecule. An early termination criteria is applied if a point is found to be out of bounds or to collide with the structure, and the thread moves on to the next grid point. Once a thread has examined all grid points within its x-slice, it updates x to the next unclaimed value and begins processing that slice. This is repeated until all slices have been processed. This dynamic schedule is preferable because structures may have a non-uniform allocation of atoms, and it might be hard to come up with a good static allocation of work across threads apriori. In general, we tried to minimize synchronizations and data sharing across multiple threads.

In addition to multithreading, we optimized the per-thread performance with SIMD extensions, and further improved performance by minimizing coordinate conversion (e.g. Cartesian x, y, z to fractional a, b, c) operations with respect to the unoptimized function. We also manually inline functions and avoid unnecessary if-else statements. Finally, we employ loop unrolling to minimize instruction dependencies.

4.6 Optimizations for GPU implementation

The GPU approach differs from the multicore implementation in that each thread on the GPU analyzes a single grid point; however, the procedure for calculating the distance and set values for each grid point is

the same, including the early termination criteria. We take special care to pack constant quantities (atom coordinates, etc.) into the GPU constant memory. This has the advantage of fast access from the GPU cores as well as reduced register pressure, enabling more threads to be executed.

Our optimized GPU kernel includes the optimizations applied to the CPU function (excluding SIMD). It consumes 36 bytes of shared memory and 18 registers. We maximize occupancy by tailoring the arrangement of threads specifically to each GPU. Finally we also use optimal functions for single-precision operations such as fmin() and rintf().

4.7 Parallelization of segmentation step

In the pseudocode provided in section 3, the flood fill algorithm segments accessible regions within some volume and finds the segments which constitute channels through analyzing their periodic boundary crossings; when a segment touches the boundary of the grid, it must be connected to the segment which is represented by the periodically adjacent grid point. Our strategy for parallelization of this routine is to first split the grid into n distinct domains, segmenting each independently on a separate thread, and then to "stitch" the resulting segments together by considering inter-domain boundaries in the same manner as the periodic boundary. The grid is split into n domains simply by considering all N grid points in order and assigning the first N/n to the first domain, and so on. As such, for comparatively large n it is possible for a domain to be a disconnected region of space, however this condition is robust under our pseudocode.

4.8 Parallelization of pocket blocking step

A pocket is a connected region of grid points which does not form a channel; it may however cross the periodic boundary, and if so is considered as more than one segment. Each segment can be blocked independently of any other, and hence our parallel implementation consists of the existing pocket blocking algorithm, with n pthreads deployed, each of which considers and blocks the first unblocked pocket in the list.

5 Results

We implemented all of the optimizations presented in the earlier section, and now present results from our experiments. Table 2 presents the absolute timings for all three steps in the algorithm for 1 to 24 Opteron cores. Timings for the cost-grid step are presented for the Fermi hardware. The number of pockets found is also presented. These results are further illustrated in Figures 3 to 8.

These results illustrate the fundamental differences between the three steps in our algorithm. The cost-grid calculation is shown (Figure 3) to continually benefit from increasing the number of threads up to 24; indeed, for all but the smallest of the examined structures, a near linear speed-up is observed with respect to the number of pthreads deployed. It is clear that this step benefits further still from the GPU implementation (Table 2); in the two larger structures, speed-ups of over 250x compared to a single CPU thread are observed, which in turn correspond to a 11x speed-up over the 24 thread implementation. These speed-up factors are lower for the smaller structures, for which it can be seen that the computation time is much smaller even on a single CPU thread.

By contrast, the time spent in segmentation step (Table 2) exhibits behavior wherein increasing the number of threads up to eight is beneficial, however beyond this point three of the four structures begin to demonstrate a reduction in performance, particularly pronounced for the smallest structure, EDI. With a large number of threads, the grid is divided into a large number of independent domains, decreasing the time spent per domain but increasing the amount of boundary crossings that must be examined. Hence, we observe (Figure 4) that a saddle exists in the region of eight threads, and increasing beyond this point, either with more pthreads or with a GPU approach, is unlikely to be beneficial in the general case. The results for pocket blocking (Figure 5) show near linear speed-up up to four threads, excluding EDI; EDI has only one pocket, and so it is clear that the multicore implementation does not yield any benefit. Since the two larger structures contain over 100 pockets, they show continued speed-up up to 24 threads, however speed-up factor with respect to the increasing number of threads can be seen to be decreasing, and hence it is expected that a GPU implementation is unlikely to be advantageous. An additional issue with respect to a GPU pocket blocking routine is that the whole cost grid must be queried while spheres are being created to ensure that the spheres do not overlap with other segments, and this would place considerable constraints on GPU memory.

We also consider overall application speed-up (Figure 6), assuming n threads applied to each step, or in the case of the GPU implementation, the maximum number of threads (24) for the segmentation and blocking steps. Since the cost-grid has already been observed to be the most computationally expensive step, the speed-ups in this portion of the algorithm dominate the overall speed-ups observed. As such, near linear speed-up is observed in the overall application for the larger structures. Further total speed-up can be achieved with the GPU implementation of the cost-calculation step; the results for Fermi indicate that on the largest structure, LTN, we achieve over a 165x speed-up over single CPU thread, and over 7x speed-up

over 24 CPU threads. Figures 7 and 8 present, respectively, the total calculation time and speedup for the LTN zeolite.

6 Discussion

While significant effort was made in optimizing the CPU performance in order to achieve a more balanced comparison with GPUs there are still additional optimizations that could be exploited. Specifically, the data for the CPU implementation is currently arranged as a Array of Structures (AoS) format rather than the more SIMD-friendly Structure of Arrays (SoA) format. The current AoS layout leaves one of the four SIMD slots unused as well as a less efficient data layout occasionally requiring more data swizzling. In addition, a SoA format would more readily exploit a wider, eight-slot SIMD unit such as those available on the latest Intel CPUs. However, the relatively short length of the innermost loop leads us to believe that the amount of efficiency gained for our current optimizations represents the majority of speed-up obtainable on a CPU. We also did not pay attention to NUMA issues on the multicore implementation by utilizing the first-touch policy on memory pages. We also did not use blocking optimizations (cache, thread, etc) on the CPU. While our code is not bandwidth bound, it is possible that both of these implementations will improve the performance of the CPU implementation.

A natural algorithmic optimization to consider is to use an acceleration structure such as an octree to improve the collision detection process. Currently, we do an $O(n)$ scan through the array of structure atoms. A tree-based structure can potentially improve the performance to $O(\log n)$. Such methods are very well suited to CPUs, but not as well to GPUs due to non-uniform memory access and thread divergence. We would like to utilize functionality from the CUDA Data Parallel Primitives Library [40] towards this end. We would like to consider other algorithmic and optimization methods in future work.

7 Conclusion

Our three-step algorithm for the analysis of the void space of porous materials has demonstrated the suitability of this algorithm for acceleration through exploitation of its parallel nature. The first step, the cost-grid calculation, has the inherently parallel nature and therefore it was implemented on multicore and GPU architectures. We have also investigated the benefits of parallelization of the other two steps in our algorithm, however we find that they do not exhibit the same inherently parallel nature as the cost-grid, and exhibit characteristics which limit the benefit of parallelization. They were implemented on multicore CPU.

Nevertheless, since the cost-grid is by far the most expensive single step in the algorithm, total application time speed-up using 24 Opteron threads for the largest structure in our dataset is over 23x (i.e. near linear), and an over 7x further speed-up in the total application is observed by utilizing the GPU for the cost-grid calculation.

Acknowledgments

The authors would like to thank Samuel W. Williams and Jihan Kim for valuable suggestions regarding code optimization. We would also like to thank NERSC for access to GPU computing facilities. This work was supported by the Director, Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231 through the Scientific Discovery through Advanced Computing (SciDAC) program's Visualization and Analytics Center for Enabling Technologies (VACET). This work was supported in part by the Applied Mathematical Science subprogram of the Office of Energy Research, U.S. Department of Energy, under Contract Number DE-AC03-76SF00098, and by the Computational Mathematics Program of the National Science Foundation.

References

1. Auerbach, S.M., Carrado, K.A. and Dutta, P. K. (2004). Handbook of Zeolite Science and Technology; Marcel Dekker: New York, USA.
2. Smit, B. and Maesen, T.L.M. (2008). *Nature* **457**:671-677.
3. Smit, B. and Maesen, T.L.M. (2008). *Chem. Rev.* **108**:4125-4184.
4. Krishna, R. and van Baten, J.M. (2007). *Chem. Eng. J.* **133**:121-131.
5. Millward, A.R. and Yaghi, O.M. (2005). *J. Am. Chem. Soc.* **127**:17998-17999.
6. Walton, K.S., Millward, A.R., Dubbeldam, D., Frost, H., Low, J.J., Yaghi, O.M. and Snurr, R.Q. (2008). *J. Am. Chem. Soc.* **130**:406-407.
7. Banerjee, R., Phan, A., Wang, B., Knobler, C., Furukawa, H., O'Keeffe, M. and Yaghi, O.M. (2008). *Science* **319**:939-94.
8. Sumida, K., Hill, M.R., Horike, S., Dailly, A. and Long, J.R. (2009). *J. Am. Chem. Soc.* **131**:15120-15121.
9. Choi, H.J., Dinca, M. and Long, J.R. (2008). *J. Am. Chem. Soc.* **130**:7848-7850.

10. Foster, M.D.; Treacy, M.M.J. http://www.hypotheticalzeolites.net (accessed Nov 13, 2010)

11. Earl, D.J. and Deem, M.W. (2006). *Ind. Eng. Chem. Res.* **45**:5449-5454.

12. Deem, M.W., Pophale, R., Cheeseman, P.A. and Earl, D.J. (2009). *J. Phys. Chem. C* **113**:21353-21360.

13. Frenkel, D. and Smit, B. (2002). Understanding molecular simulations. 2nd ed.; Academic Press, San Diego, USA, pp. 23-62.

14. (a) Dubbeldam, D. and Smit, B. (2003). J. Phys. Chem. B. **107**:12138.; (b) Bates, S.P., v. Well, W.J.M., v. Santen, R.A. and Smit, B. (1996). *J. Am. Chem. Soc.* **118**:6753.

15. Krishna, R. and van Baten, J.M. (2010). *Langmuir* **26**:2975-2978.

16. Haranczyk, M. and Sethian, J.A. (2010). *J. Chem. Theory Comput.* **6**:3472-3480.

17. Haranczyk, M. and Sethian, J.A. (2009). *Proc. Natl. Acad. Sci. USA (PNAS).* **106**:21472-21477.

18. Keffer, D., Gupta, V., Kim, D., Lenz, E., Davis, H. T. and McCormick, A. V. (1996). *J. Mol. Graph.* **14**:108-116.

19. Theisen, K., Smit, B. and Haranczyk, M. (2010). *J. Chem. Inf. Model.* **50**:461-469.

20. First, E.L.; Gounaris, C.E.; Wei, J.; Floudas, C.A. (2011). *Phys. Chem. Chem. Phys.* **13**:17339-17358.

21. Till, M.S. and Ullmann, G.M. (2010). *J. Mol. Model.* **16**:419-429.

22. Petrek, M., Otyepka, M., Banas, P., Kosinova, P., Koca, J. and Damborsky, J. (2006). *BMC Bioinformatics* **7**:316-325.

23. Foster, M.D., Rivin, I., Treacy, M.M.J. and Friedrichs, O.D. (2006). *Microporous and mesoporous materials* **90**:32-38.

24. Blatov, V.A., Ilyushin, G.D., Blatova, O.A., Anurova, N.A., Ivanov-Schits, A.K. and Dem'yanets, L.N. (2006). *Acta Cryst.* **B62**:1010-1018.

25. Haldoupis, E., Nair, S. and Sholl, D.S. (2010). *J. Am. Chem. Soc.* **132**:7528-7539.

26. Willems, T.F.; Rycroft, C.H.; Kazi, M.; Meza, J.C.; Haranczyk, M. (2011) *Microporous and mesoporous materials* doi:10.1016/j.micromeso.2011.08.020 .

27. Ufimtsev, I.S.; Martinez, T.J. (2008). *J. Chem. Theory Comput.* **4**:222-231.

28. Ufimtsev, I.S.; Martinez, T.J. (2009). *J. Chem. Theory Comput.* **5**:1004-1015.

29. Ufimtsev, I.S.; Martinez, T.J. (2009). *J. Chem. Theory Comput.* **5**:2619-2628.

30. Stone, J.E.; Hardy, D.J.; Ufimtsev, I.S.; Schulten, K. (2010).*J. Mol. Graph. Model.* **29**:116-125.

31. a) Baumes, L.A.; Kruger, F.; Jimenez, S.; Collet, P.; Corma, A. (2011). *Phys. Chem. Chem. Phys.* **13**:46744678; b) Krger, F.; Maitre, O.; Jimnez, S.; Baumes, L.A.; Collet, P. Speedups between 70 and 120 for a generic local search (memetic) algorithm on a single GPGPU chip, Vol. 6024, 2010.
32. Jiang, J.; Jorda, J.L.; Yu, J.; Baumes, L.A.; Mugnaioli, E.; Diaz-Cabanas, J.-M.; Kolb, U.; Corma,A. (2011). *Science* **333**:1131-1134.
33. a) Maitre, O.; Lachiche, N.; Clauss, P.; Baumes, L.A.; Corma, A.; Collet, P. (2009) Efficient parallel implementation of evolutionary algorithms on GPGPU cards, Vol. 5704, 2009; b)O. Maitre, L. A. Baumes, N. Lachiche, A. Corma, P. Collet, in 11th Annual Genetic and Evolutionary Computation Conference, GECCO-2009, Montreal, 2009.
34. Sethian, J.A. (1996). *Proc. Nat. Acad. Sci. (PNAS).* **93**:1591-1595.
35. Sethian, J.A. (1999). Level Set Methods and Fast Marching Methods, Cambridge University Press, 2nd Edition, New York, pp. 86-99.
36. Dijkstra, E.W. (1959). *Numerische Mathematic* **1**:269-271.
37. CUDA Developer Zone. http://www.nvidia.com/object/cuda_home_new.html.
38. GPU Computing. http://www.nvidia.com/object/GPU_Computing.html.
39. http://www.gpgpu.org/.
40. CUDA Data Parallel Primitives Library. http://code.google.com/p/cudpp/.
41. Jeong, W.; Whitaker, R.T. (2007). A Fast Eikonal Equation Solver for Parallel Systems. In SIAM Conference on Computational Science and Engineering.

8 Author's Biographies

Richard Luis Martin is a Postdoctoral Research Fellow in the Scientific Computing Group at Lawrence Berkeley National Laboratory. He received his PhD in Cheminformatics and BSc in Computer Science and Mathematics from The University of Sheffield. His research interests include the development of algorithms and descriptors for classification and screening.

Prabhat is a member of the Visualization group at Berkeley Lab. Prabhat received an MS in Computer Science from Brown University in 2001 and a B.Tech in Computer Science and Engineering from IIT-Delhi in 1999. His current research interests include Scientific Visualization, High Performance Computing, Parallel I/O, GPGPU, Machine Learning and Applied Statistics.

David Donofrio is a member of the Advanced Technologies Group at Berkeley Lab. David received his degree in Computer Engineering from Virginia Tech in 2001 and was previously a member of Intel's 3D Graphics Architecture team. His research interests include: energy efficient computing, computer architecture, high performance computing and performance modeling.

James A. Sethian is Professor of Mathematics at the University of California, Berkeley, and Head of the Mathematics Department at the Lawrence Berkeley National Laboratory.

Maciej Haranczyk is a Research Scientist in the Scientific Computing Group at Berkeley Lab. Dr. Haranczyk received a PhD and MS degrees in Chemistry from University of Gdansk, Poland. He spent his post-doctoral appointment as a 2008 Glenn T. Seaborg Fellow at Berkeley Lab. His research interests include development of methods, tools and approaches to enable efficient molecular and materials discovery.

Core Architecture	**AMD Opteron**	**NVIDIA GF100**
Type	out-of-order SMD	in-order SIMT
Clock (GHz)	2.1	1.15
SP GFlop/s	8.4	73.6
L1 Data Cache	128K	16 KB
L2 Data Cache	512K	48 KB
SMP Architecture	**Opteron 6172 Magny-Cours**	**Tesla C2050 (Fermi)**
Threads per core	1	4096
Cores per socket	12	14†
Sockets per SMP	2	1
Shared Last-level cache	12MB	768 KB
Aggregate System DRAM	48GB	3GB
Aggregate DRAM GB/s	80	144 (no ecc)
Aggregate SP GFlop/s	100.8	1030.4

Table 1. Details of the evaluated architectures. †For simplicity, we call each shared multiprocessor (SM) on a GPU a "core". All bandwidths and flop rates are peak theoretical.

Table 2. Detailed timing results for various materials and hardware platforms

		ABSOLUTE TIME (seconds, including copying time for GPU)				***SPEED-UP OVER 1 CORE***			
Structure	Implementation	Cost-grid	Flood Fill	Pocket Blocking	*Total application*	Cost-grid	Flood Fill	Pocket Blocking	*Total application*
LTN	1 core	2122.761	1.063	78.404	*2202.355*	1.000	1.000	1.000	*1.000*
(152 pockets)	2 cores	1056.060	0.591	37.609	*1094.387*	2.010	1.799	2.085	*2.012*
	4 cores	530.136	0.338	18.557	*549.159*	4.004	3.145	4.225	*4.010*
	8 cores	265.455	0.253	9.635	*275.471*	7.997	4.202	8.137	*7.995*
	16 cores	135.404	0.288	5.469	*141.290*	15.677	3.691	14.336	*15.587*
	24 cores	89.137	0.340	4.843	*94.446*	23.815	3.126	16.189	*23.319*
	Fermi GPU	7.915	(0.340)	(4.843)	*13.224*	268.193	(3.126)	(16.189)	*166.542*
TSC	1 core	853.992	23.541	125.924	*1003.547*	1.000	1.000	1.000	*1.000*
(116 pockets)	2 cores	426.948	7.887	62.107	*497.030*	2.000	2.985	2.028	*2.019*
	4 cores	214.443	5.071	30.866	*250.470*	3.982	4.642	4.080	*4.007*
	8 cores	108.776	4.553	16.558	*129.977*	7.851	5.170	7.605	*7.721*
	16 cores	55.834	4.370	8.971	*69.262*	15.295	5.387	14.037	*14.489*
	24 cores	37.893	4.186	7.346	*49.513*	22.537	5.624	17.142	*20.268*
	Fermi GPU	3.336	(4.186)	(7.346)	*14.956*	255.971	(5.624)	(17.142)	*67.099*
HEU	1 core	5.490	0.075	0.994	*6.586*	1.000	1.000	1.000	*1.000*
(26 pockets)	2 cores	2.707	0.055	0.503	*3.279*	2.028	1.364	1.976	*2.009*
	4 cores	1.332	0.055	0.254	*1.654*	4.122	1.364	3.913	*3.983*
	8 cores	0.673	0.056	0.186	*0.927*	8.158	1.339	5.344	*7.105*
	16 cores	0.344	0.058	0.186	*0.601*	15.968	1.293	5.344	*10.962*
	24 cores	0.235	0.071	0.189	*0.507*	23.374	1.056	5.259	*12.993*
	Fermi GPU	0.114	(0.071)	(0.189)	*0.386*	47.955	(1.056)	(5.259)	*17.041*
EDI	1 core	0.114	0.010	0.038	*0.170*	1.000	1.000	1.000	*1.000*
(1 pocket)	2 cores	0.058	0.005	0.034	*0.105*	1.948	2.000	1.118	*1.611*
	4 cores	0.035	0.005	0.036	*0.083*	3.279	2.000	1.056	*2.053*
	8 cores	0.025	0.005	0.048	*0.085*	4.644	2.000	0.792	*2.010*
	16 cores	0.016	0.007	0.042	*0.070*	7.099	1.429	0.905	*2.425*
	24 cores	0.013	0.012	0.047	*0.078*	9.077	0.833	0.809	*2.191*
	Fermi GPU	0.009	(0.012)	(0.047)	*0.074*	13.196	(0.833)	(0.809)	*2.307*

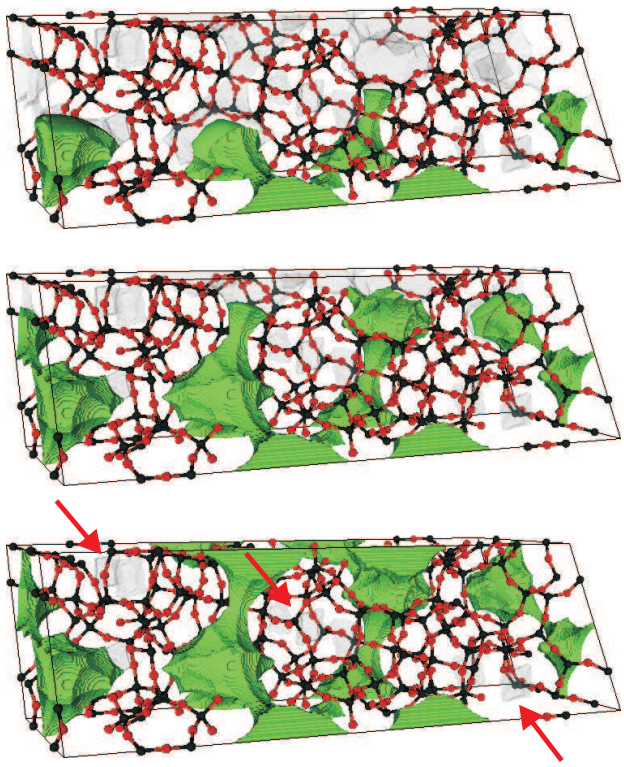

Fig. 1. These snapshots illustrate a propagating front (in green) that explores void channels in the periodic unit cell of DDR zeolite. Red arrows point to inaccessible pockets that cannot be accessed by the front.

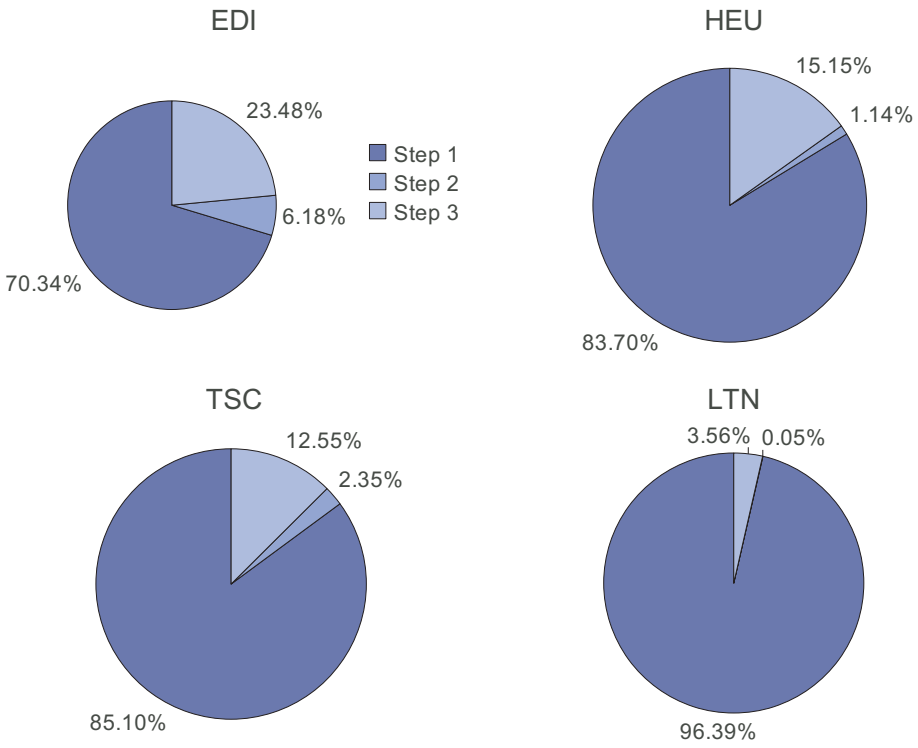

Fig. 2. Profile of execution time on various Zeolite structures.

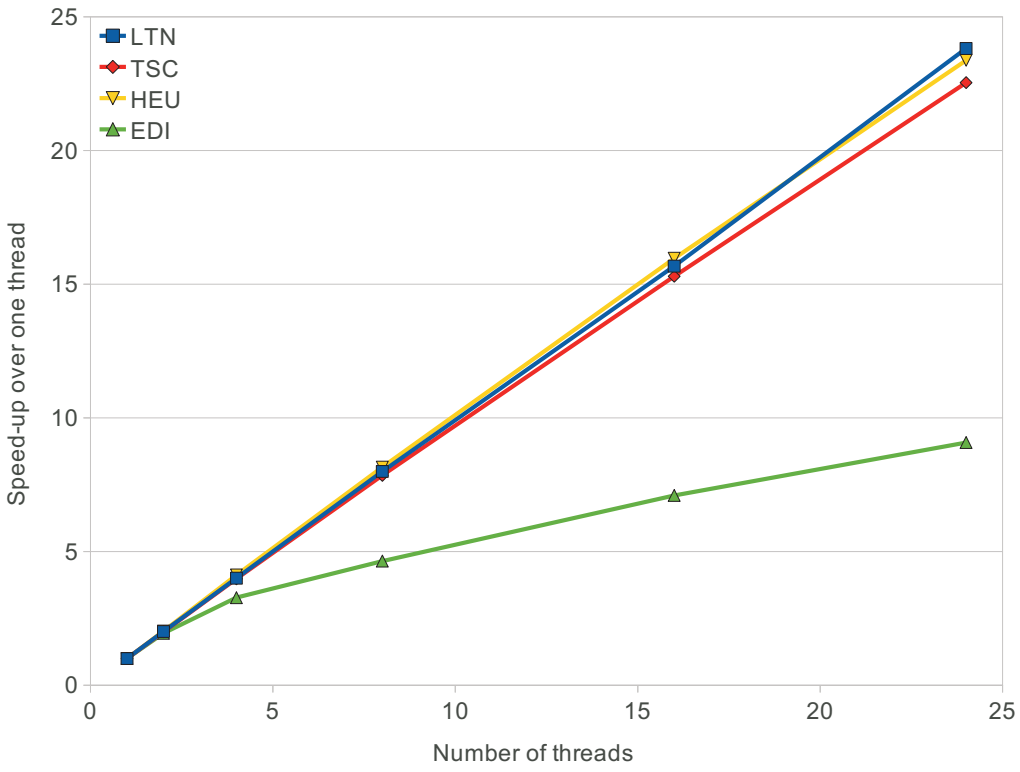

Fig. 3. Speedup results for cost-grid calcluation on multicore CPU.

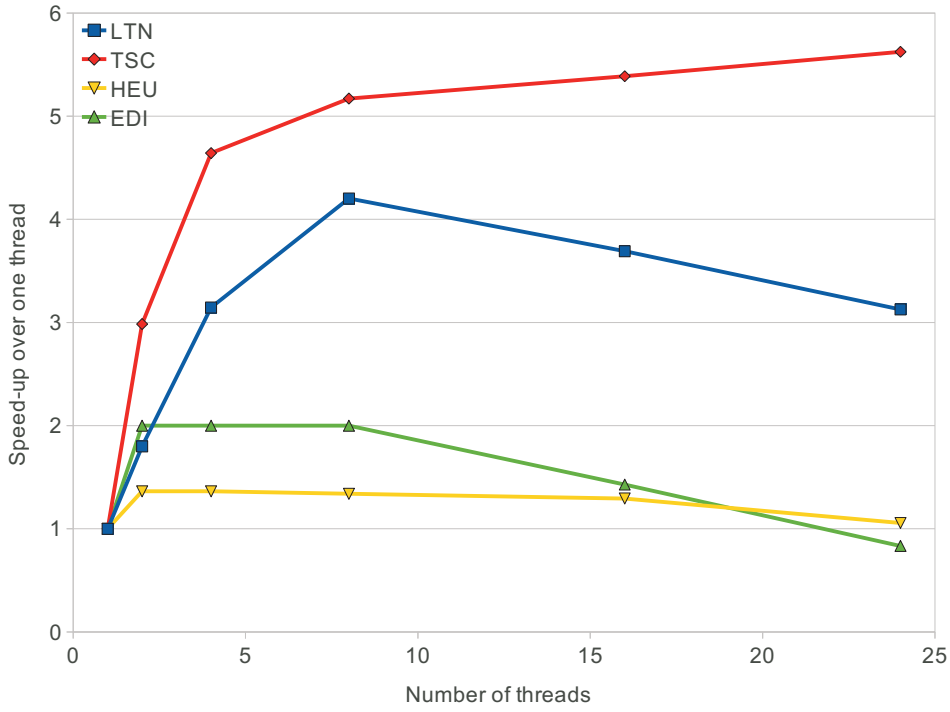

Fig. 4. Speedup results for flood fill calculation on multicore CPU.

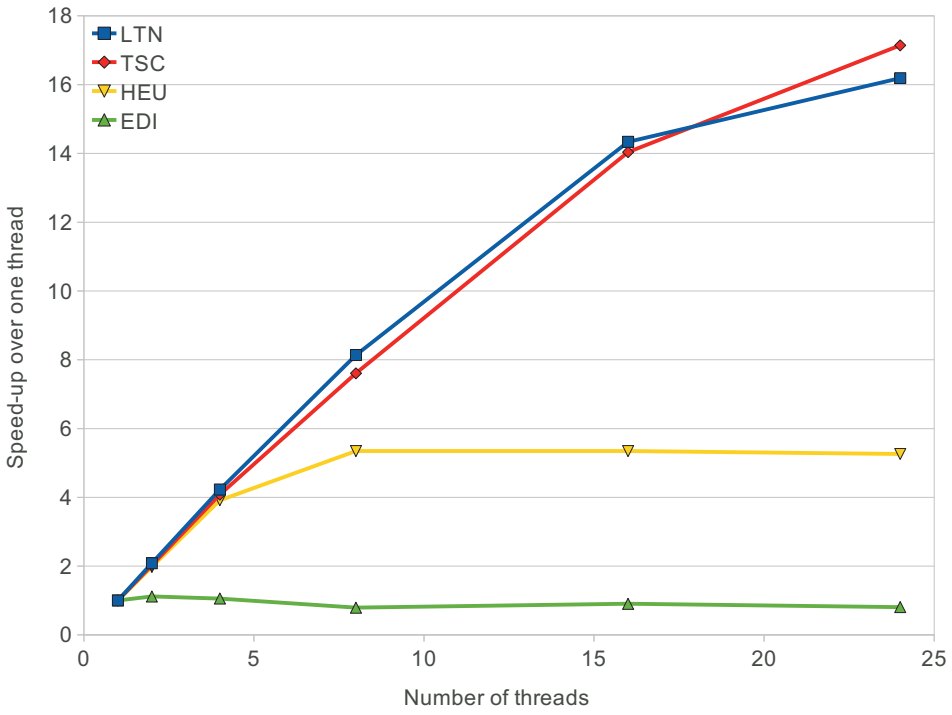

Fig. 5. Speedup results for pocket blocking calculation on multicore CPU.

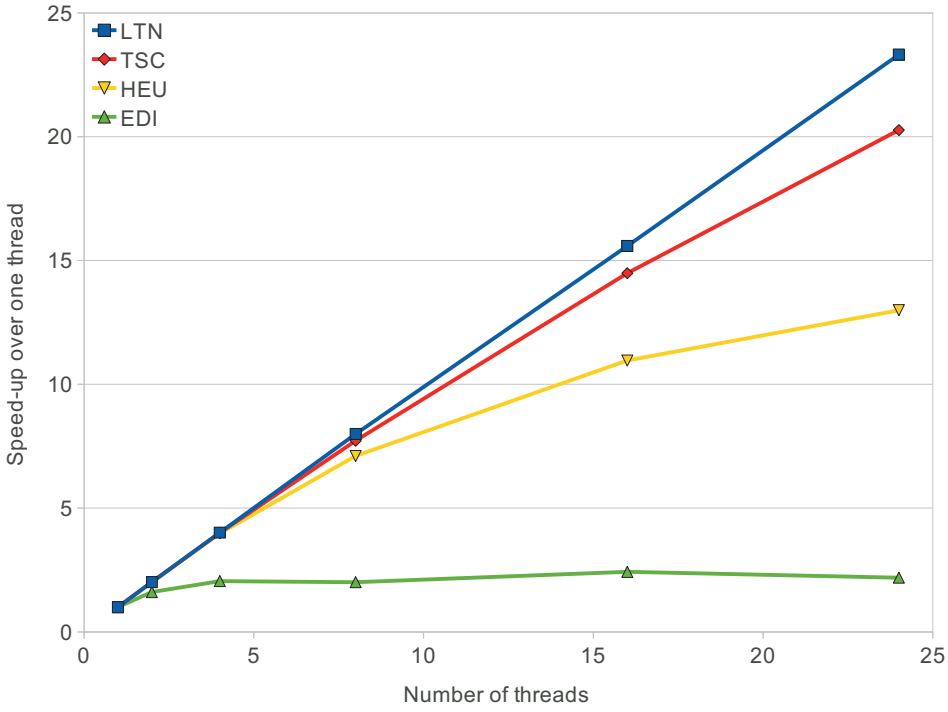

Fig. 6. Speedup results for entire application executed on multicore CPU.

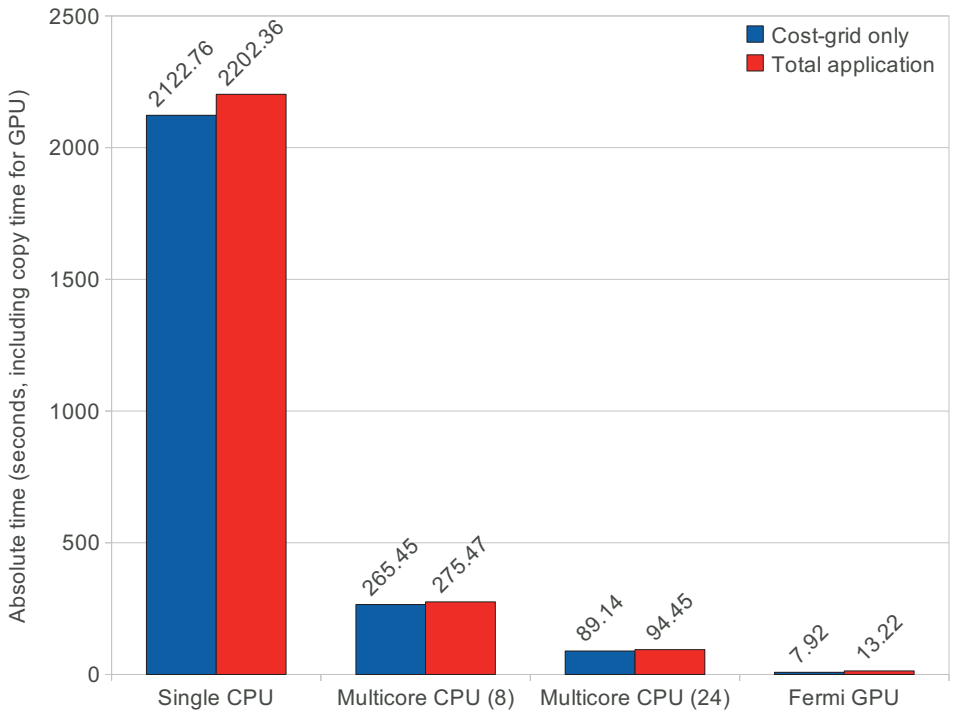

Fig. 7. Absolute times for computation of LTN zeolite on multiple platforms.

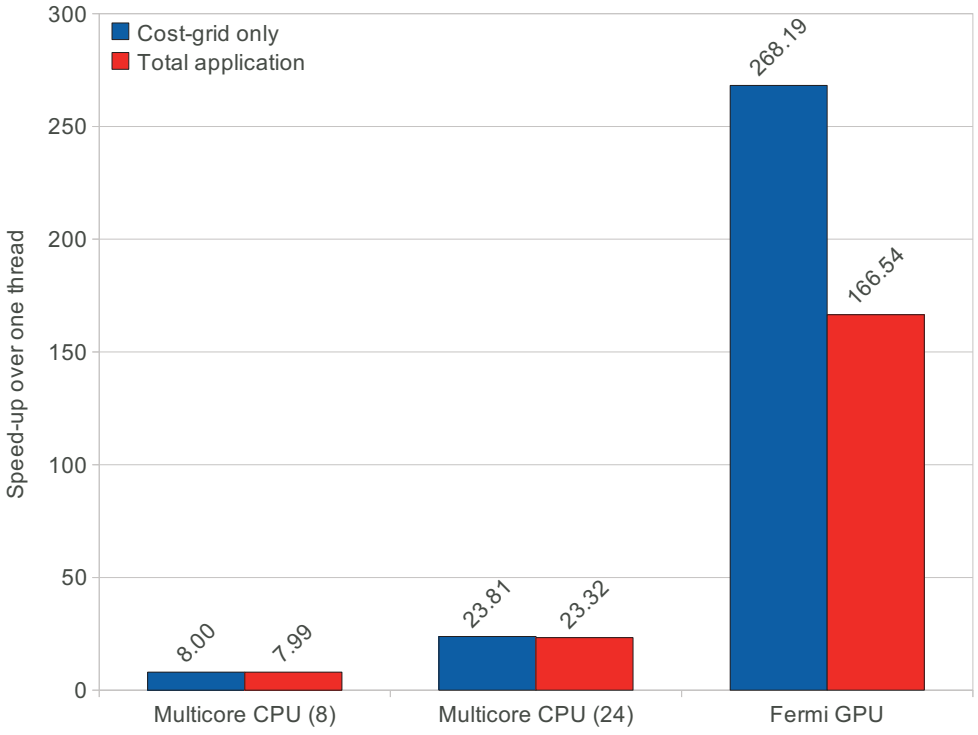

Fig. 8. Speedup results for computation of LTN zeolite on multiple platforms.