ANL/ALCF/ESP-13/5

Cosmic Structure Probes of the Dark Universe (Porting and Tuning HACC on Mira)

ALCF-2 Early Science Program Technical Report

Argonne Leadership Computing Facility

ANL/ALCF/ESP-13/5

Cosmic Structure Probes of the Dark Universe (Porting and Tuning HACC on Mira)

ALCF-2 Early Science Program Technical Report

prepared by
Hal Finkel
Argonne Leadership Computing Facility, Argonne National Laboratory

May 7, 2013

Porting and Tuning HACC on Mira

Hal Finkel

April 4, 2013

Abstract

The HACC (Hybrid/Hardware Accelerated Cosmology Code) framework has achieved over 69% of the floating-point peak running on all of Mira, and is now uniquely positioned to make significant contributions to the science of large-scale structure formation. At the beginning of the Early Science Project (ESP), HACC was attaining under 1% of the peak FLOPS, and lacked the memory-management and file I/O capabilities necessary for running at scale. This report describes the improvements that transformed HACC from an average scientific code to one well-known for its unmatched performance.

1 Introduction

Modern cosmology, and the study of large-scale structure formation in particular, is now a precision science, with both simulation and observation contributing to our overall understanding of a vast quantity of observational data. As the quantity and quality of observational data increases, corresponding increases in our simulation ability are required. In order to tackle questions related to the detailed behavior of dark matter and dark energy, the two dominate, yet mysterious, components of the energy-density of the universe [2, 1], we must be able to simulate the effect of small changes in theory parameters on observable statistical quantities.

After the emission of what is now the Cosmic Microwave Background (CMB) during recombination, the non-linear and long-range nature of the gravitational force becomes increasingly important to the overall evolution of the matter in the universe. At scales larger than that of galaxy clusters, no other forces make a significant contribution[1]. As a result, we can accurately predict how the matter will move and cluster at large scales using a "gravity-only" simulation. Because of the huge dynamic range required almost everywhere in the simulation volume, almost all structure-formation codes use a particle method whereby the matter in the universe is decomposed into many point masses which are evolved using a modified form of Newton's equations. The HACC (Hybrid/Hardware

[1]Assuming that the effect of dark energy on the background evolution of the universe is taken into account.

Accelerated Cosmology Code) framework provides an infrastructure for running just these kinds of simulations and performing some of the analysis necessary to compare the simulated universes to observational data.

Prior to the beginning of the Early Science Project (ESP), large simulations had been run with HACC only on the Roadrunner machine at LANL. The largest of these simulations used a few hundreds of billions of particles, and made use of the PowerXCell 8i GPU-like processors in order to quickly compute the short-range particle-particle force contributions [5]. HACC's ability to run on an massively-parallel CPU-only system like Mira was very limited, and initial tested showed that it achieved under 1% of the peak FLOPS (floating-point operations per second). In addition, its required one-file-per-rank I/O facility would have been impractical for use at scale, and its poor memory management would have precluded long runs. Over the course of the last two years, all of these things have greatly improved. In fact, HACC was a 2012 finalist for the IEEE/ACM Gordon Bell prize in scientific high-performance computing [4], reporting 13.94 PFlops at 69.2% of peak and 90% parallel efficiency on 1,572,864 cores running on the BG/Q machine Sequoia at LLNL. To meet today's science requirements, we'll need to run trillions of particles in several large simulations: HACC is now ready for that challenge!

2 Long-Range Force

The total gravitational force on every particle from every other particle is split into two parts: the long-range part and the short-range part. The long-range force is computed on a grid using a FFT (Fast Fourier Transform)-based technique. Unfortunately, memory limits the size of the grid to a resolution near the initial inter-particle spacing. As the universe evolves, and matter clusters together under gravity, the relevant length scale for particle-particle interactions in clustered regions falls well below the grid resolution. Adding this additional short-range component is the responsibility of the architecture-specific short-range force solver. Nevertheless, the long-range force solver, and specifically the routines used to compute the distributed FFTs on which the solver relies, determine the overall weak scaling of the code. As shown in Figure 1, the overall weak scaling of the code is nearly perfect, across architectures, and specifically on nearly all of Mira.

In order to achieve this scaling on tens of thousands of ranks, the algorithm using to compute the FFTs was changed. Prior to the ESP, HACC used a slab-decomposed FFT, meaning that the grid is decomposed in only one dimension. Unfortunately, this limits the total number of ranks that can participate in the FFT computation (because there can be only as many ranks as there are grid points in the smallest dimension). Due mostly to work by Nicholas Frontiere, this limitation was removed by implementing a distributed slab-decomposed FFT. The slab decomposed FFT enables as many ranks to participate as there are points on the smallest two-dimensional face. This difference is illustrated in Figure 2. Once completed, this slab-decomposed FFT allowed HACC to easily

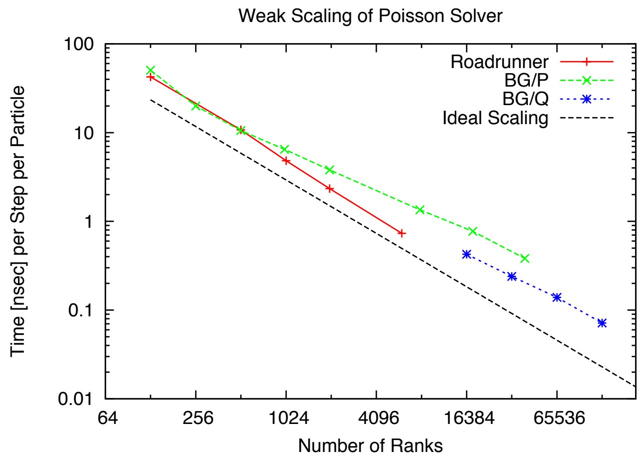

Figure 1: Scaling of the long-range force solver.

scale to the largest-available machines.

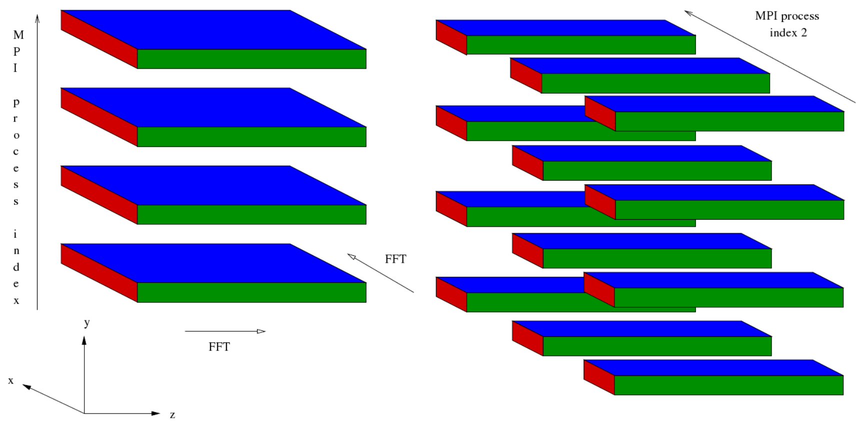

Figure 2: A diagram showing slab (left) and pencil (right) data decompositions [derived from diagrams in [6]].

It is worth mentioning that a significant reason that HACC's inter-rank communication needs are dominated by the FFT, and not by "ghost-region" particle exchange, is because the exchange of particles between ranks happens relatively infrequently. Not only is the short-range force computation subcycled (computed multiple times per long-range force computation), but the ghost-region exchanges do not even need to happen prior to every long-range computation. Compared to the size of the universe, the velocities of matter in the universe are relatively small, and the size of densely-clustered regions is limited. As a result, we can make the ghost regions large enough to capture whatever large structures might reside on neighboring ranks close to the inter-rank boundary, and we can safely assume that they won't move very far in between ghost-region exchanges. To mitigate edge effects, we also don't compute the short-range force on particles near the outer boundary of the ghost region.

3 Short-Range Force and the RCB Tree

The short-range force is determined by taking the complete gravitational force between any two particles, and subtracting that part of the force that will be captured by the long-range force solver. This subtraction is determined by performing numerical experiments, and then the resulting functional form is created to fit the results of these experiments. Prior to the ESP, HACC used a lookup table to compute the force from its fitting form. Now, HACC uses a 5^{th}-order polynomial fit to the effective short-range part of the force. Specifically, this is:

$$f_{SR}(s) = (s + \epsilon)^{-3/2} - f_{grid}(s) \tag{1}$$

where $s = \mathbf{r} \cdot \mathbf{r}$, $f_{grid}(s) = \text{poly}[5](s)$, and ϵ is a short-distance cutoff.

While on Roadrunner (and on machines with GPUs) HACC can use a statically-partitioned N^2 algorithm to compute the local particle forces, such a naive algorithm would be too slow for Mira's CPUs. Instead, we must use a short-range force computation scheme which effectively approximates the force from farther-away particles on any given particle and only directly computes the contributions from the closest particles. The standard way of accomplishing this feat in an efficient way is to use a "tree" code. This means that some spatial-partitioning tree is used along with a monopole approximation for forces from particles in far-away tree nodes. Before the ESP, HACC has an octree-based tree code, but that implementation was inefficient, both computationally and in terms of memory overhead. Addressing these problems required a new design, and I implemented what has been called in the physics literature a Recursive Coordinate Bisection (RCB) tree[2]. Following the suggestion in [3], a RCB tree recursively splits each spatial node on its longest side such that the dividing plane intersects the center of mass of the particles in the node. As also suggested in [3], when each node is partitioned, the particle data for that node is also partitioned, and the particles are shuffled into the two subpartitions. Finally, this splitting process stops when the nodes reach a certain maximum number of particles. These implementation details, illustrated in Figure 3, have important performance benefits: The center-of-mass splitting makes the computation in each leaf node balanced by insuring that each leaf node has near the maximum number of allowed particles. The partitioning insures that each node's particle data exhibits a high degree of cache locality. Finally, keeping multiple particles in each leaf node gives us a tuning parameter allowing us to

[2]A computer scientist would, however, call it a KD-tree.

trade a fast but asymptotically expensive operation (the particle-particle interactions, which scale as N^2) for a slow but asymptotically favorable operation (the tree walk, which has $N \log N$ scaling). An additional benefit to keeping multiple particles in each leaf node is an increase in the accuracy of the computation. As can be seen from Figure 4, even making aggressive use of the monopole approximation still yields highly-accurate force computations in clustered regions when many particles are contained in each leaf node. On Mira, we normally run with a few hundred particles in each leaf node, and the resulting error from the monopole approximation is quite small.

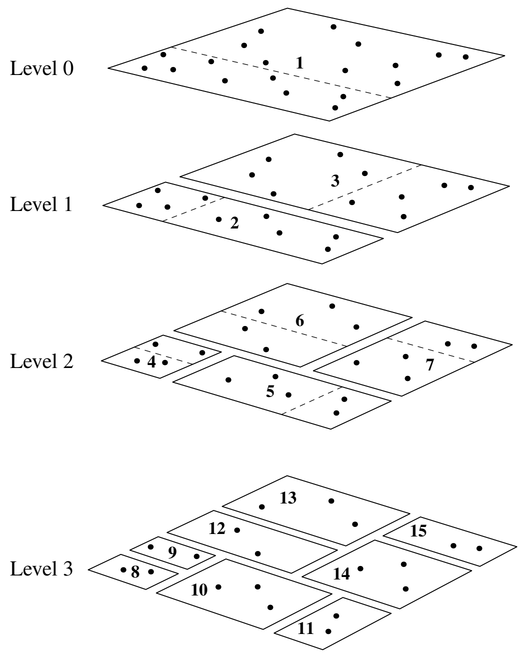

Figure 3: A diagram (from [3]) showing how the particles are recursively partitioned into RCB-tree nodes.

Another critical aspect of the attaining good performance on Mira is how multiple threads per rank are used to compute the short-range force on each particle. At this point, two different schemes have been used in production. The first scheme, which was used in the runs used for the Gordon Bell submission, confines the threading to the force computation within each RCB-tree leaf node. Within each leaf node, the particles are divided among the available threads, and because all particles in a leaf node share the same interaction list, each thread simply computes the force on its particles from those particles in the interaction list. The downsides of this scheme are that the per-leaf-node concurrency is limited by the number of particles per leaf node (which limits scaling), and that the process of walking the tree to compose the interaction list is not threaded. As a result, this scheme could scale to only 8 threads per rank. To overcome this

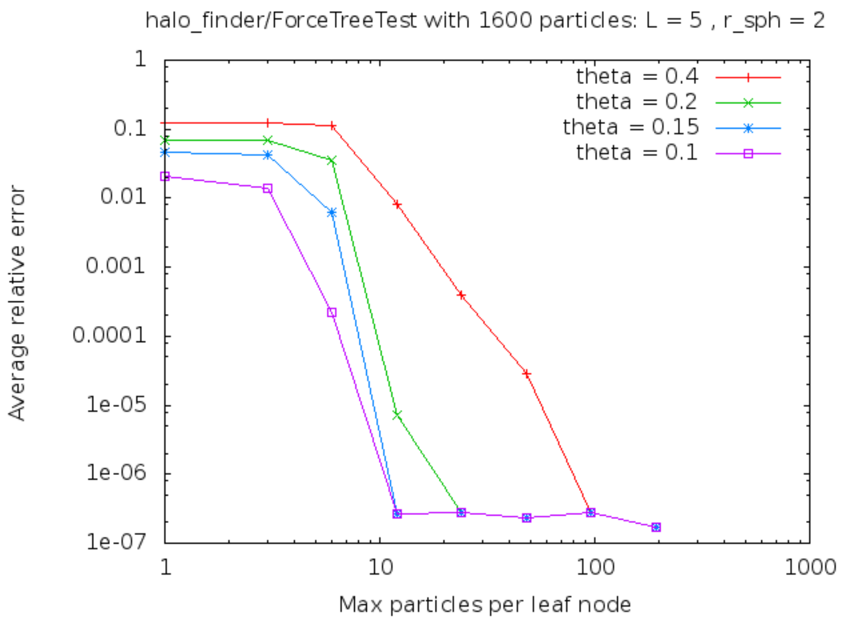

Figure 4: A plot of a force-computation test involving a sphere randomly filled with particles showing the relative error from the monopole approximation. The theta parameter controls the aggressiveness of the monopole approximation.

limitation, I implemented a second scheme. In this second scheme, a queue is composed of all leaf nodes, and these leaf nodes are divided among all available threads for processing (both the tree walk to form the interaction list and the force computation itself). As there are many leaf nodes per rank, as shown in Figure 5, this scheme can scale to 32 thread per rank on Mira. OpenMP was used to implement both schemes.

4 The Short-Range Force Kernel

The BG/Q-specific short-range force kernel, implemented by Vitali Morozov, uses QPX intrinsic functions, and a loop which is manually partially unrolled, in order to quickly evaluate the force on some particle by other particles in the interaction list. While the kernel appears to be a straightforward vectorization of the loop over particles in the interaction list (see Figure 6 for an exerpt), the form of the force computation (for example, the $-3/2$ exponent and the order of the polynomial in Equation 1) were chosen to map well onto the available instructions and the number of available registers.

Several other aspects of the force kernel implementation are worth noting: First, as shown in Figure 7, all data loaded by the kernel in explicitly prefetched

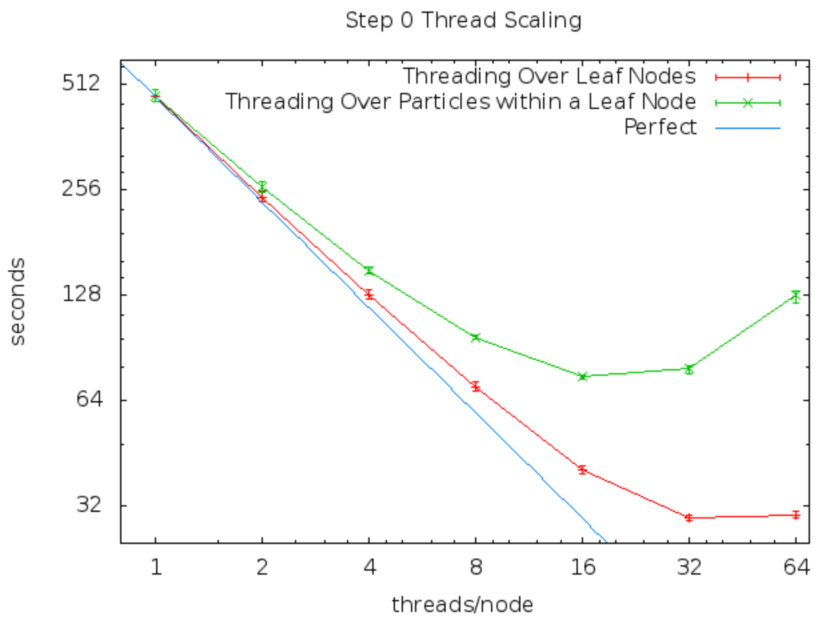

Figure 5: A plot of the time taken by the short-range force computation vs. the number of threads. The old threading scheme could not scale past 8 threads per rank, while the new scheme can scale to 32 threads per rank (running one rank per node).

into the L1 cache. On the BG/Q, even when the L1 prefetcher is prefetching a stream of interest, the data is not put into the L1, but rather stays in a separate L1P buffer, until accessed. Because the L1P access latency is much higher than that associated with accessing data from the L1 cache, explicitly prefetching the data into the L1 cache is critical. Second, we don't need fully IEEE-compliant values for the square root in Equation 1, and it can be computed using the provided reciprocal square root estimate and one Newton-Raphson iteration for refinement. Because the provided estimate is accurate to within one part in 2^{14}, only one iteration is required to provide the single-precision result. Finally, as branches are expensive, we use the provided "select" intrinsic in order to avoid branching as part of the distance filtering in the force computation. These intrinsics are shown in Figure 8. The force kernel itself runs as well over 80% of the available peak FLOPS.

5 Memory management

Pushing the simulation limits of large-scale structure formation means running simulations with as many particles as possible, and this implies that we're con-

```
for ( i = 0, j = 0; i < count1−7; i = i + 8, j = j + 32 )
{
    ...
    b0 = vec_sub( b0, a1 );
    c0 = vec_sub( c0, a1 );

    b0 = vec_mul( b0, b0 );
    c0 = vec_mul( c0, c0 );

    b1 = vec_ld( j, yy1 );
    c1 = vec_ld( j+16, yy1 );
    ...
    b1 = vec_madd( b2, a15, a14 );
    c1 = vec_madd( c2, a15, a14 );

    b1 = vec_madd( b2, b1, a13 );
    c1 = vec_madd( c2, c1, a13 );

    b1 = vec_madd( b2, b1, a12 );
    c1 = vec_madd( c2, c1, a12 );
    ...
```

Figure 6: An excerpt from the short-range force kernel showing some QPX intrinsics and the basic form of the partially-unrolled loop. The polynomial force evaluation makes use of many fused multiply-add instructions.

```
for ( i = 0, j = 0; i < count1−7; i = i + 8, j = j + 32 )
{
    __dcbt( (void *)&xx1 [i+offset] );
    __dcbt( (void *)&yy1 [i+offset] );
    __dcbt( (void *)&zz1 [i+offset] );
    __dcbt( (void *)&mass1[i+offset] );
    ...
```

Figure 7: An excerpt from the short-range force kernel showing that all interaction-list data is prefetched into the L1 cache.

stantly running as close as possible to the memory limit of the machine. As a result, memory fragmentation [3] becomes a large problem. To make matters worse, HACC is required to allocate and free different data structures during different parts of each time step because there is not enough available memory to hold all such structures at the same time. Furthermore, many of these data structures, such as the RCB tree used for the short-range force calculation, have

[3]Memory fragmentation refers to the condition where small allocations dispersed throughout the memory space leave no large contiguous chunks free even though the total amount of free memory may be large.

```
for ( i = 0, j = 0; i < count1−7; i = i + 8, j = j + 32 )
{
    ...
    b1 = vec_rsqrte( b0 );
    c1 = vec_rsqrte( c0 );
    ...
    b0 = vec_sel( b1, a6, b2 );
    c0 = vec_sel( c1, a6, c2 );
    ...
```

Figure 8: An excerpt from the short-range force kernel showing the reciprocal square root estimate and "select" intrinsics.

sizes that change dynamically with each new time step. This, combined with other allocations from the MPI implementation, message printing, file I/O, etc. with lifetimes that might outlast a time-step phase, is a recipe for fatal memory fragmentation problems. Indeed, our first at-scale runs on Mira were limited in duration precisely because of this problem. To mitigate this problem I implemented a specialized pool allocator called Bigchunk. As the name implies, this allocator grabs a large chunk of memory, and then distributes it to various other subsystems. During the first time step, Bigchunk acts only as a wrapper of the system's memory allocator, except that it keeps track of the total amount of memory used during each phase of the time step. Before the second time step begins, Bigchunk allocates an amount of memory equal to the maximum used during any phase of the previous time step plus some safety factor. Subsequent allocations are satisfied using memory from the big chunk, and all such memory is internally marked as free after the completion of each time-step phase. This en-mass deallocation design implies that the bigchunk allocator has minimal overhead, and the time it takes to allocate memory from Bigchunk is very small compared to the speed of the system allocator. Because the Bigchunk memory is not released back to the system, the memory fragmentation problem has effectively been solved.

6 I/O

Prior to the ESP, the initial conditions for a HACC simulation were produced by a stand-alone (parallel) program. This program would produce one input file per rank, which was the only input mode supported by HACC at the time. While the process of producing the initial conditions is fairly fast, the requirement to write hundreds of thousands, or millions, of initial-conditions files, and then read them in again would have been prohibitive. As a result, I integrated the initializer into the HACC simulation code, and enhanced it to use the pencil-decomposed FFT discussed above.

Once HACC could generate its own initial conditions, the next challenge was making sure it could write out particle snapshots for analysis in a reasonable amount of time (meaning under approximately 30 minutes). After initial experiments using single-file MPI-I/O proved unable to meet our performance requirements, in collaboration with Venkat Vishwanath, I designed and implemented a new file I/O framework, called Generic I/O, capable of meeting our needs. Several important features are:

- For small files, using collective MPI-I/O is necessary, but for large writes using non-collective MPI-I/O or POSIX I/O is necessary. Using the Generic I/O framework, we can choose between any of these three options. On the largest scales, using POSIX I/O yields the best performance on Mira.
- The total output file size is pre-calculated before the writes begin, and the file extent is pre-allocated. This reduces locking contention on the parent directory's metadata.
- To avoid locking contention, we write one file per BG/Q I/O node. There is one I/O node per 128 compute nodes on Mira.
- Each rank writes into a disjoint space (without any kind of data reorganization).
- The file format is self-describing, and in addition to ASCII converters, I created a plugin for the SQLite database engine to make querying data in the files straightforward and flexible.
- 64-bit CRC (Cyclic Redundancy Check) "checksum" codes are stored covering all data and metadata, where each variable from each rank has its own CRC code. This allows the data to be validated before it is used for a simulations checkpoint restart or for analysis.

To the last point regarding CRC codes, although on-disk data corruption is rare, and undetected corruption of data while traversing the network is perhaps rarer still, it can still be a significant problem at scale. As a point of reference, a relatively-small simulation recently run at NERSC created 100 checkpoint files, each one TB is size. Among all of those files, in one file, one variable from one rank was corrupted. This corruption was detected when the checkpoint was read in, and the restart was aborted. Had we not protected the data with CRC codes, we might have not noticed, until many CPU hours had been wasted, that the simulation was partially invalid, and it might have been impossible to later determine why. With the CRC-code protection, the user was immediately alerted to the problem, and was able to restart from an earlier checkpoint. In short, I recommend that all groups make sure that there data is protected by CRC codes.

7 Conclusion

Putting together all of the pieces discussed here has resulted in an astounding improvement of HACC's performance and scalability. HACC is now well positioned to make significant contributions to the science of large-scale structure formation. In addition, with additional processing, HACC's large simulated universes can be used as test beds for the analysis software used on real observational data. None of this would have been possible without the dedicated effort of (in no particular order) Salman Habib, Vitali Morozov, Adrian Pope, Katrin Heitmann, Venkat Vishwanath, Nicholas Frontiere, and many others. Working with all of these people has been a great pleasure, and is a good example of how multi-disciplinary collaborations can yield truly-significant results.

References

[1] Weinberg, D.H., M.J. Mortonson, D.J. Eisenstein, C. Hirata, A.G. Riess and E. Rozo, arXiv:1201.2434 [astro-ph.CO].

[2] Frieman, J.A., M.S. Turner, and D. Huterer, Ann. Rev. Astron. & Astrophys. **46**, 385 (2008).

[3] Gafton, E. and S. Rosswog, Mon. Not. R. Astron. Soc., to appear (2011), arXiv:1108.0028v1

[4] Habib, S., et al., arXiv:1211.4864 [cs.DC].

[5] Habib, S., et al., J. Phys.: Conf. Ser. 180 012019 (2009).

[6] Scott, D. M. "Scaling Turbulence Applications to Thousands of Cores: a dCSE Project" (2010)