Parallelising HiPSTAR using OpenMP

Tom Edwards (Cray Centre of Excellence for HECToR), Richard Sandberg (University of Southampton)

HiPSTAR is a computational fluid dynamics application for simulating turbulence and noise generated by flow. In a current EPSRC funded project, HiPSTAR is used to investigate noise generated by turbulent jets exiting aircraft engines. The full scientific aims and merits of this application is described in detail in an earlier report to the Cray Centre of Excellence (CoE) Steering Committee[1].

This document summarises the software engineering effort by the CoE to transform HiPSTAR from a pure MPI parallel application to one that uses MPI and OpenMP in a hybrid parallel application. Originally HiPSTAR was a distributed memory application that had the data domain decomposed along two of the three spatial dimensions. MPI messages were passed between processors to allow communication and the application scales well to thousands of cores. However, as the application scales to the much larger core counts available on modern Cray XE supercomputers the number of data points assigned to each core approaches the minimum allowed by the core algorithms. Yet, as the number of points in the distributed dimensions reduces, the number of points in the third non-parallel dimension remains constant.

To enable HiPSTAR to continue scaling, the code should be adapted to allow data in the third dimension to be processed in parallel by multiple cores. This requires a hybrid MPI and OpenMP approach because data is frequently transformed along the third dimension between the physical and Fourier domains using Fast Fourier Transforms (FFTs). Each FFT is a global operator requiring data from all points along the dimension which could potentially cause a communications bottleneck when using MPI alone. A shared memory approach would alleviate this. However, previous generations of the architecture have provided two or four shared memory cores per node. With the recent upgrade of the HECToR service to the Cray XT6 then XE6 supercomputer the number of shared memory cores has increased to 24 cores per node which allows much greater scope for using shared memory parallelism.

Parallelising the third dimension with OpenMP is expected to improve HiPSTAR's performance in at least two ways:

1. As hybrid application HiPSTAR will be able to scale further than the pure MPI version as additional parallelism is extracted from the third dimension
2. Using hybrid MPI+OpenMP is expected to be more efficient than the pure MPI. Using shared memory reduces the overhead of storing and processing replicated "halo" data from ranks on the same shared memory node as the original data is accessible. Reducing the number of MPI ranks reduces the number of small messages exchanged between nodes which are replaced by fewer, larger, messages that are more efficient.

The approach to Hybridising HiPSTAR

OpenMP provides many constructs for sharing work between threads in individual loops. However, there is an inevitable cost in performance when synchronising all the threads after each parallel region which is exacerbated

[1] (http://www.hector.ac.uk/coe/pdf/HiPSTAR_FinalReport_July2010.pdf)

by having large numbers of parallel regions. To achieve effective parallel speed up, synchronisation must be limited to only necessary points and significant proportions of application's run time must be in parallel. This rules out individual loop level parallelism with OpenMP as an impractical solution for HiPSTAR.

Instead the data domain is decomposed at a much higher level, in a similar manner to the distributed memory MPI component. Each thread is assigned a sub-domain along the third dimension that it processes, only synchronising with other threads when necessary. As the default is to run in parallel, sections of code that are inherently serial or non-thread safe are explicitly trapped and performed by a single "master" thread. This high level approach to parallelism ensures that a significant proportion of the application runs in parallel, enough to ensure a substantial speed up.

Changes to HiPSTAR's Code

The largest change to HiPSTAR was replacing all the bounds of all loops over the third dimension with extents that divided the entire range into unique sub-sections for each thread. This change was applied programmatically across the vast majority of the code leaving only the more isolated and complicated cases for hand alterations.

Sections that are not thread safe or are best performed by a single thread, like Network Communication MPI or File I/O operations are performed in serial by a single thread. The intrinsic OpenMP master, single and barrier operations were not used directly because they cause deadlocks when called from within a parent "master" region. Instead "smart" master and barrier routines were created that were aware of being called from nested structures and prevent deadlocks in the application. These routines can also dynamically change the extent of loops over the third dimension, allowing the same code to be called in parallel or from a master thread and produce the same result.

The FFTs required by the application are only performed in one dimension, so rather than parallelising the individual FFT calls, the application takes advantage of the independence of each call from the others. Each transformation from or to the Fourier domain synchronises all threads then divides the Fourier transforms between the threads before synchronising and continuing. This is easy to implement with shared memory as each thread has access to all memory so there is no change to the layout of data in memory. To perform the same action using MPI would require communication between processors and transposing the original or a copy of data which ultimately degrades the application's performance.

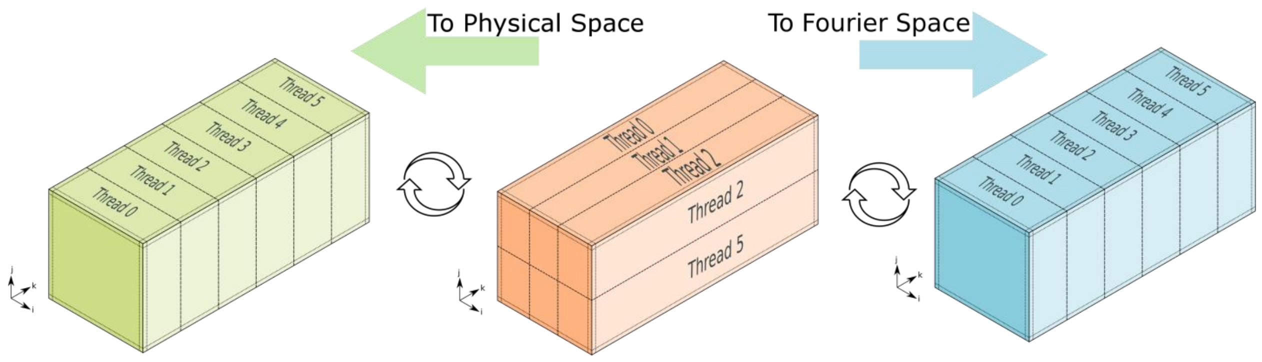

Figure 1 Transposing threads to perform the Fourier Transform

Evaluating the Performance

The hybrid code has been tested on the HECToR's Cray XE6, comparing the pure MPI version with the hybrid. Each version ran on the same total number of cores, either with an individual MPI rank per core or with one MPI rank decomposed over 6 threads (the internal structure of the Opteron Magny-Cours processor favours 4x6 threads compared to 1x24 threads). Runs ranged from 384 (16 nodes) cores to 24 576 (1024 nodes) all with the same problem size (strong scaling).

The first result was that the pure MPI task did not complete when running on 24 576 cores. The hybrid version ran to completion demonstrating that decomposing along the extra dimension allows the application to scale to larger core counts. Figure 1 shows that the hybrid version has better scaling and continues to scale further than the original pure MPI.

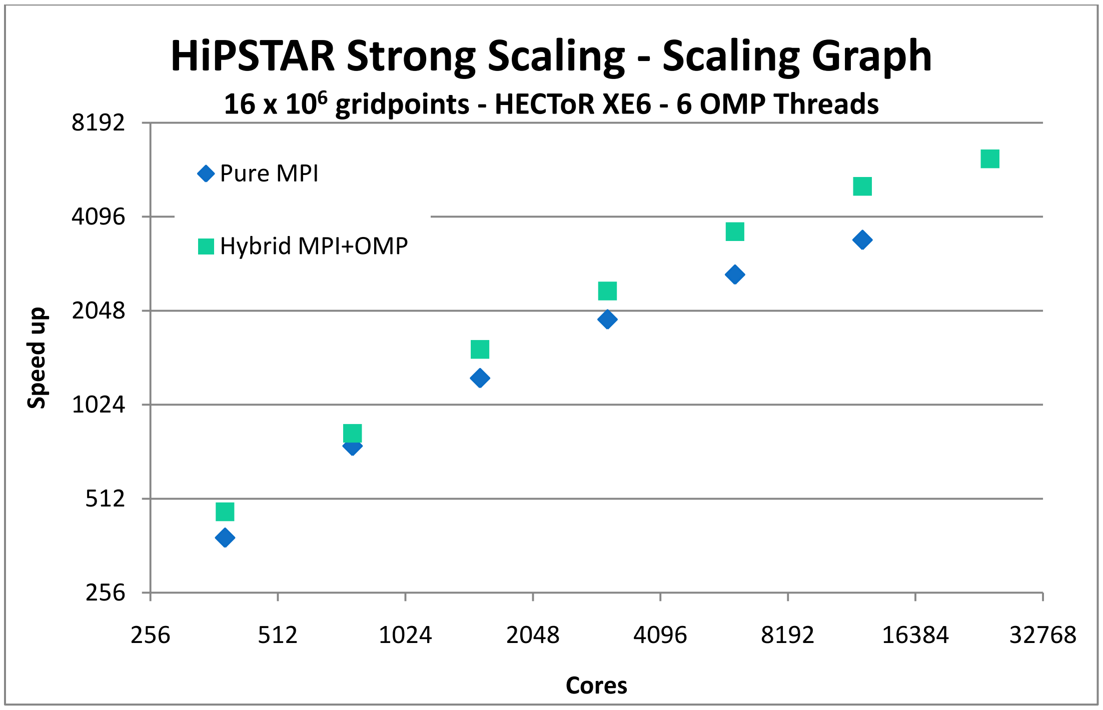

Figure 2 HiPSTAR Scaling Graph

Figure 2 shows the total computational cost on all processors for each time step. It shows that for all core counts the hybrid code runs more quickly on the same total number of cores resulting in a faster time to solution. The improvements range between 10% - 50% gains, with, importantly, the greatest gains at the largest core counts.

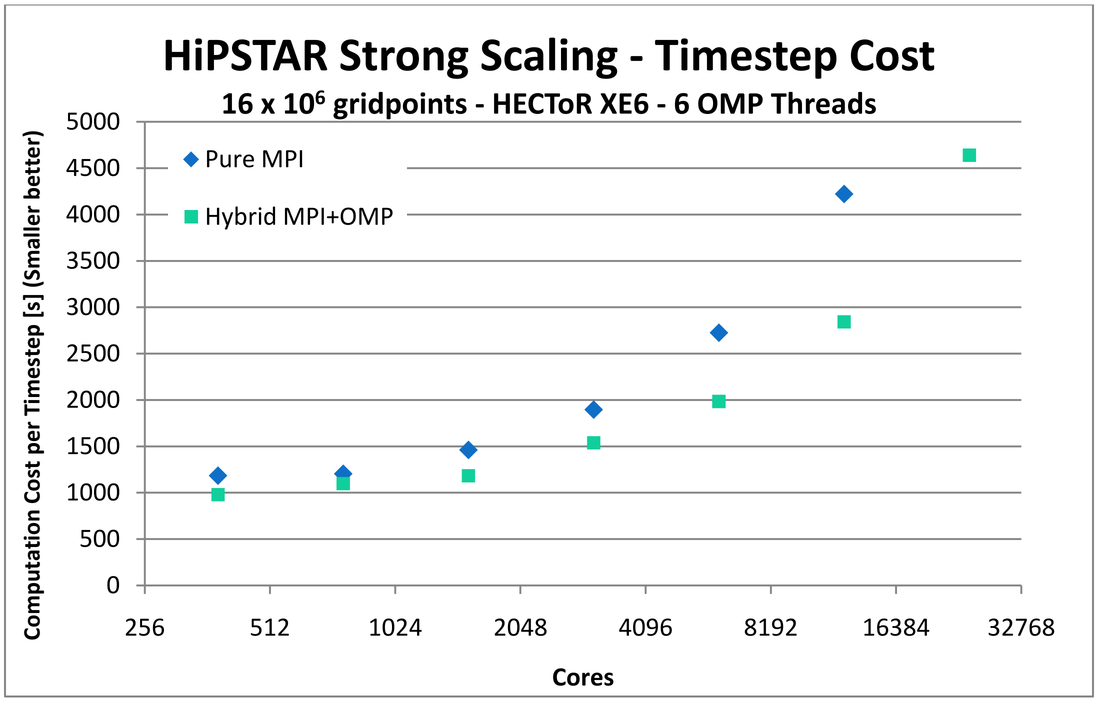

Figure 3 HiPSTAR Timestep Cost Graph