Parallel Algorithms (15-499), Spring 09
Assignment #5 **Due: April 9th**

Problem 1: OpenMP
In this problem, you will revisit the merge-sort and the matrix transpose code of assignment 3, and parallelize it using OpenMP instead of Cilk++. OpenMP is similar to Cilk++ in that it consists of a set of directives that the progammer inserts in to C++ code to exploit parallelism in the code. However, unlike in the case of Cilk++ where having nested `cilk_spawn` commands in recursive functions did not hurt the performance significantly, you need to be more careful about spawning off too many threads in OpenMP. There are several ways to control the number of threads in OpenMP, the exact commands depending on the compiler chosen for compiling. You can use the Sun compiler (with the following flags) on the `multi6` machine:

```
/opt/sun/sunstudio12/bin/CC -xopenmp -xO5 -Qpath /usr/bin/ -o <exec> <prog.cpp>
```

You can find the user guide at:

`http://docs.sun.com/app/docs/doc/819-5270;`

section 4 on nested parallelism might be of some help. A more basic tutorial can be found at:

`http://www.nic.uoregon.edu/iwomp2005/iwomp2005_tutorial_openmp_rvdp.pdf.`

You should run your code on the `multi6` machine. Run mergesort with a data set of size $10,000,000$ and your matrix transpose with a matrix of size 7000×7000. Compare these times with the corresponding times for the sequential code (don't forget to compile with `-xO5` compiler optimization flag). Submit your code by copying in to the handin directory:

`/afs/cs.cmu.edu/academic/class/15499-s09/handin/username/hw5/`

Problem 2: Closest pair in 3-dimensions

Given a finite set of points $X \subseteq \mathbb{R}^d$ and a Euclidean distance metric $d(\cdot,\cdot)$, the problem of finding the closest pair is to report a pair of points that are closest together. As seen in lectures, on n two-dimensional points, a simple divide-and-conquer algorithm can solve this problem in work $O(n \log n)$ and depth $O(\log^2 n)$. You may wonder what can be said about higher dimensions.

It is known that for n d-dimensional points, an extension of the divide-and-conquer algorithm can solve the closest-pair problem in work $O(n \log^{d-1} n)$ and depth $O(\log^d n)$. In this problem, you will explore the 3-d case: design and analyze an algorithm for the 3-d closest-pair problem with work $O(n \log^2 n)$ and depth $O(\log^3 n)$.

(**Hint:** If you use the same outline as the 2-d version, when you get to finding the closest pair within the 2δ "slap," try to reduce it to the 2-d case. It will not be exactly the 2-d problem we saw in class, but it should be pretty similar.)