Logistics

- NERSC accounts were set up this week.
- Anyone get their password from NERSC yet?
  - See the notes link for homework #3
- Passwords
  - All initial passwords must be changed for proper security!
  - Three character classes and 1 special character are required for the password change
  - Changing the password at NERSC takes 1 hour to propagate to all nodes!
    - Don’t wait until the last minute!!!!!!!!!
- Homework #3 is posted and assigned.
  - The supporting links are very important.
  - Links to compiler documentation etc.
  - The OpenMP Pi code is there.
Logistics [2]

Are there any Questions??
Isoefficiency

- Efficiency is a function of the problem size and the number of processors
  - For a fixed sized problem
    - Efficiency decreases as the number of processors increase
    - Efficiency increases as the problem size increases
  - The goal is to figure out how to scale the problem size and the number of processors with a constant efficiency!
Efficiency as a function of processor count

Efficiency

Number of Processors
Efficiency as a function of Problem Size

Efficiency

Problem Size
Isoefficiency a function of Problem Size and processor count

Efficiency

Problem Size and Processor Count
### Efficiency of adding n numbers in on p processors

<table>
<thead>
<tr>
<th></th>
<th>p=1</th>
<th>p=4</th>
<th>p=8</th>
<th>p=16</th>
<th>p=32</th>
</tr>
</thead>
<tbody>
<tr>
<td>n=64</td>
<td>1.0</td>
<td>.80</td>
<td>.57</td>
<td>.33</td>
<td>.17</td>
</tr>
<tr>
<td>n=192</td>
<td>1.0</td>
<td>.92</td>
<td>.80</td>
<td>.60</td>
<td>.38</td>
</tr>
<tr>
<td>n=320</td>
<td>1.0</td>
<td>.95</td>
<td>.87</td>
<td>.71</td>
<td>.50</td>
</tr>
<tr>
<td>n=512</td>
<td>1.0</td>
<td>.97</td>
<td>.91</td>
<td>.80</td>
<td>.62</td>
</tr>
</tbody>
</table>
Isoefficiency Metric

- **Quinn’s derivation is valid** see section 7.6
- **The Result is:**
  - $T(n,1)$ is the sequential execution time
    - Is proportional to the time for all processes doing work not done by the sequential algorithm.
  - $T_o(n,p)$ is the time for all processes doing work not done by the sequential algorithm including overhead components.
    - $T_o(n,p) = (p-1)\sigma(n) + p\kappa(n,p)$
  - The proportionality constant is a function of the efficiency (what we want to be constant!)
    - $C = \varepsilon(n,p) / [1 - \varepsilon(n,p)]$
  - $T(n,1) = C \cdot T_o(n,p)$
Isoefficiency Metric

- $T(n,1) = C \cdot T_0(n,p)$
  - The serial work as a function of the problem size ($n$) has to scale more than the extra serial work and the overhead as a function of problem size and processor count.
  - A small isoefficiency function (e.g., $O(1)$) means that it takes only a small increase in problem size to maintain a constant efficiency
    - A scalable system
  - A large isoefficiency function (e.g., $O(n^4)$) means that a large problem size is needed to maintain constant efficiency!
    - Scalability is limited and may also be limited by other resources (e.g., memory)
Example

- **Summing** $n$ numbers on $p$ processors.
  - Split among $p$ processors $n/p$ local work per thread
  - Sequential work is $O(n)$
  - Communications can be done in $p \log p$ time.

- $n \geq C \cdot p \log p$ or $C \leq n/(p \log p)$

- This means that $n$ must increase. How?
  - $C = C$ and $n' = kn$
  - $n/(p \log p) = n'/(p' \log p') = kn/(p' \log p')$
  - $k = (p' \log p')/(p \log p)$
OpenMP Compilers

• The intel compiler is on osage and redwing
  ➢ See homework #3 details.
  ➢ You ARE expected to be able to read documentation and figure out how to use the compiler.

• ANY other compiler you want to use for development purposes is fine as long as:
  ➢ It supports the OpenMP standard
  ➢ It does not introduce extensions to the standard that you use!

• The homework requirement is that the code works on the IBM SP at NERSC!
On the SCL Intel Systems

• **Compilers are in /opt/intel/compiler70**
  - Source the file:
    - /opt/intel/compiler70/ia32/bin/iccvars.csh
    - Or for bash use /opt/intel/compiler70/ia32/bin/iccvars.sh
  - Then icc (intel C/C++ Compiler) should be in your path.
    - Might have to issue rehash for tcsh.
  - Links to documentation are on the homework page
  - Many optimization flags you should determine which ones improve the serial performance of your code!
On the SCL Intel Systems [2]

- **Access to systems**
  - First go to isugw.scl.ameslab.gov (ssh only)
  - From there ssh to osage or redwing
    - Osage and redwing share home directories for student accounts.
    - Run interactively 😊
    - Osage 2Ghz Xeons 1Mb cache
      - 8 GB memory
      - 4 real processors
        ✌️ Thinks it has 8 processors
        ✌️ Hyperthreading is turned on
    - Redwing 2Ghz Xeons 2Mb cache
      - 16 GB memory
      - 4 processors
Stupid pet tricks with GNU make [1]

CC = ls -l
FC = ls -l
_OS = $(shell uname )

ifeq ($(_OS),Linux)
CC = icc
CFLAGS = -O3 -OTHERFLAGS
LDFLAGS =
endif

ifeq ($(_OS),AIX)
CC = xlc
CFLAGS = -O -OtherFlags
LDFLAGS = -bmaxdata:0x50000000 -bmaxstack:0x10000000
endif
Stupid pet tricks with GNU make [2]

OBJ = matrix.o gen_A.o gen_B.o gen_C.o print_mat.o set_it.o \ mat_mul.o mat_mul_daxpy.o mat_mul_blocked.o \ mat_mul_blocked_daxpy.o vec_compare.o Analyze_matrix.o \ CPU_Time.o Wall_Time.o Usage.o vec_norm.o \ gen_Ap.o gen_Bp.o gen_Cp.o

matrix: $(OBJ)
    $(CC) $(LDFLAGS) $(CFLAGS) -o $@ $(OBJ)
# all the dependencies are deleted from this slide but you need them here
# for include files etc.

clean:
    -rm -f $(OBJ) work.pcl work.pc
realclean:
    -rm -f core core.* matrix
    $(MAKE) clean
Stupid pet tricks with GNU make [1]

CC = ls -l
FC = ls -l
_OS = $(shell uname )

ifeq ($(_OS),Linux)
CC = icc
CFLAGS = -O3 -OTHERFLAGS
LDFLAGS =
endif

ifeq ($(_OS),AIX)
CC = xlc
CFLAGS = -O -OtherFlags
LDFLAGS = -bmaxdata:0x50000000 -bmaxstack:0x10000000
endif
Modules on the NERSC system

- **Read about modules on the NERSC home page.**
  - Quite often access to HPC resources means that you use more than one site.
    - You have to know and understand the environment at each site.
- **The module command adds things to your path.**
  - `% module load GNU`
    - Adds cvs, gmake etc.
  - `% module avail`
    - Tells you what modules are available.
  - `% lots of tools available.`
    - Most are not needed
  - **You should add module load commands to your .login.ext file for csh or tcsh.**
Protecting Pragma with CPP

```c
#if defined _OPENMP
#pragma omp parallel for private(comp_count,j) \
        shared(buffer,a,b,c,iend,jend)
#endif

for (i=ilo;i<iend;i++) {
    for (j=jlo;j<jend;j++) {
        comp_count = (i-ilo)*(jend - jlo) + j-jlo;
        buffer[comp_count] = a*(double)i +b*(double)j + c;
    }
}
Make sure OpenMP is used

#if defined _OPENMP
    printf("Homework #3 \n Group 17 \n Ricky A Kendall\n");
#else
    printf("Homework #3 \n NO OPENMP enabled\n");
    printf("Group 17 \n Ricky A Kendall\n");
#endif
## Timings on IBM SP: OpenMP code

<table>
<thead>
<tr>
<th>rowsA</th>
<th>colsA</th>
<th>colsB</th>
<th>threads</th>
<th>GenC</th>
<th>DDOT</th>
<th>DAXPY</th>
<th>BLOCKED+DDOT</th>
<th>BLOCKED+DAXPY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1</td>
<td>17.44</td>
<td>124.83</td>
<td>10.69</td>
<td>8.49</td>
<td>8.91</td>
</tr>
<tr>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>4</td>
<td>4.36</td>
<td>31.39</td>
<td>2.75</td>
<td>2.72</td>
<td>2.87</td>
</tr>
<tr>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>8</td>
<td>2.18</td>
<td>15.69</td>
<td>1.42</td>
<td>1.38</td>
<td>1.43</td>
</tr>
<tr>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>16</td>
<td>1.11</td>
<td>7.96</td>
<td>0.84</td>
<td>0.7</td>
<td>0.73</td>
</tr>
<tr>
<td>1500</td>
<td>1501</td>
<td>1499</td>
<td>1</td>
<td>58.77</td>
<td>450.71</td>
<td>41.45</td>
<td>30.46</td>
<td>29.25</td>
</tr>
<tr>
<td>1500</td>
<td>1501</td>
<td>1499</td>
<td>4</td>
<td>14.73</td>
<td>112.83</td>
<td>10.62</td>
<td>8.12</td>
<td>7.81</td>
</tr>
<tr>
<td>1500</td>
<td>1501</td>
<td>1499</td>
<td>8</td>
<td>7.4</td>
<td>56.63</td>
<td>5.68</td>
<td>4.88</td>
<td>4.71</td>
</tr>
<tr>
<td>1500</td>
<td>1501</td>
<td>1499</td>
<td>16</td>
<td>3.74</td>
<td>28.43</td>
<td>4.36</td>
<td>3.27</td>
<td>3.16</td>
</tr>
<tr>
<td>879</td>
<td>1257</td>
<td>912</td>
<td>1</td>
<td>17.56</td>
<td>128.84</td>
<td>11.23</td>
<td>8.77</td>
<td>8.8</td>
</tr>
<tr>
<td>879</td>
<td>1257</td>
<td>912</td>
<td>4</td>
<td>4.4</td>
<td>32.34</td>
<td>2.88</td>
<td>2.41</td>
<td>2.42</td>
</tr>
<tr>
<td>879</td>
<td>1257</td>
<td>912</td>
<td>8</td>
<td>2.2</td>
<td>16.25</td>
<td>1.56</td>
<td>1.62</td>
<td>1.62</td>
</tr>
<tr>
<td>879</td>
<td>1257</td>
<td>912</td>
<td>16</td>
<td>1.12</td>
<td>8.18</td>
<td>0.9</td>
<td>0.82</td>
<td>0.82</td>
</tr>
<tr>
<td>1001</td>
<td>489</td>
<td>1500</td>
<td>1</td>
<td>12.86</td>
<td>86.3</td>
<td>7.12</td>
<td>6.35</td>
<td>6.34</td>
</tr>
<tr>
<td>1001</td>
<td>489</td>
<td>1500</td>
<td>4</td>
<td>3.23</td>
<td>21.82</td>
<td>1.87</td>
<td>2.05</td>
<td>2.03</td>
</tr>
<tr>
<td>1001</td>
<td>489</td>
<td>1500</td>
<td>8</td>
<td>1.62</td>
<td>10.94</td>
<td>0.98</td>
<td>1.04</td>
<td>1.03</td>
</tr>
<tr>
<td>1001</td>
<td>489</td>
<td>1500</td>
<td>16</td>
<td>0.82</td>
<td>5.5</td>
<td>0.54</td>
<td>0.52</td>
<td>0.52</td>
</tr>
</tbody>
</table>
Time to Solution (1500,1501,1499)

- GenC
- DDOT
- DAXPY
- BLOCKED+DDOT
- BLOCKED+DAXPY

Seconds vs. Threads

0 4 8 12 16
Time to Solution (1500,1501,1499)

- GenC
- DDOT
- DAXPY
- BLOCKED+DDOT
- BLOCKED+DAXPY
SPEEDUP (1000,1000,1000)

- ideal
- GenC
- DDOT
- DAXPY
- BLOCKED+DDOT
- BLOCKED+DAXPY

Threads: 0, 4, 8, 12, 16
Speed Up: 0, 4, 8, 12, 16
How do you predict Timings?

- **DDOT is O(N^3)**
  - IBM SP 500,500,500 time is 13.95 seconds
  - What is the time for 1000,1000,1000?
    - $13.95 \times (1000/500)^3 = 13.95 \times 2^3 = 13.95 \times 8 = 111.6$ seconds
    - Actual time is 124.83 seconds
  - What is time for 1500,1501,1499?
    - $13.95 \times (1500/500)(1501/500)(1499/500) = 376.65$
    - Actual time is 450.71 seconds
    - Or
    - $124.83\times(1500/1000)(1501/1000)(1499/1000) = 421.30$
  - Accurate to 10% (over or under).
    - $(124.83/111.6) = 1.12$
    - $(450.71/421.30) = 1.07$
What Timings should I set

- `@ wall_clock_limit= 00:35:00`
  - 35 minutes was plenty of time to run
  - The largest case
    - 1500, 1500, 1500
  - With ALL 4 threads settings
    - `% setenv OMP_NUM_THREADS 1` /matrix
    - `% setenv OMP_NUM_THREADS 4` /matrix
    - `% setenv OMP_NUM_THREADS 8` /matrix
    - `% setenv OMP_NUM_THREADS 16` /matrix
  - Don’t submit a job larger than 40 minutes of wall time.
    - Help us preserve our allocation.
LoadLeveler Script

```bash
#@ job_name = myjob
#@ account_no = repo_name                 #our repo name is mp359
#@ output = myjob.out
#@ error = myjob.err
#@ job_type = parallel
#@ environment = COPY_ALL
#@ notification = complete
#@ network.MPI = csss,not_shared,us
#@ node_usage = not_shared
#@ class = low                               # use "regular" or "low"
#@ shell = /usr/bin/csh                     # use your favorite shell
#@ tasks_per_node = 1                      # for openmp this is 1
#@ node = 1                                 # for openmp this is 1
#@ wall_clock_limit= 00:05:00
#
#@ queue

setenv MYENVAR value
./your_binary_name
```