Tuning and Analysis Utilities

- Performance evaluation tool
- Support for multiple parallel programming paradigms: MPI, Multi-threading, Hybrid (MPI+Threads)
- Access to hardware counters.
- Automatically instruments your code.
How to use TAU?

- Set a couple of environment variables
  - $PATH, $TAU_MAKEFILE, $TAU_OPTIONS
- Instrument the program by inserting TAU macros or automatically.
- To take advantage of TAU's automatic instrumentation features, Program Database Toolkit (PDT) needs to be installed <pdt-dir>. PDT provides access to the high-level interface of source code for analysis tools and applications.
- For **automatic** instrumentation
  - Replace the compiler with TAU compiler script.
TAU Configuration

- Each configuration labeled with the options used.
  
  ```
  ./configure -mpi -arch=bgl -pdt=<pdt-dir> -pdt=xIC
  -PROFILE(default) /-PROFILECALLPATH/-MPITRACE/…
  ```

- Each configuration creates a unique Makefile.
  - `<tau-dir>/bgl/lib` for BG/L platform
  - `<tau-dir>/bgp/lib` for BG/P platform

- TAU compiler scripts are installed in
  - `<tau-dir>/ppc64/<bin>`

- Add the bin directory to your path.

  ```
  export PATH=/gpfs/home1/tulin/TAUL/tau-2.17.3/ppc64/bin:$PATH
  export PATH=/gpfs/home1/tulin/TAUP/tau-2.17.3/ppc64/bin:$PATH
  ```
Set TAU_MAKEFILE

- Set the environment variable TAU_MAKEFILE to the location of the tau makefile.
- List of TAU’s Makefile
  - Makefile.tau-mpi-pdt
  - Makefile.tau-callpath-mpi-pdt
  - Makefile.tau-mpi-pdt-mpitrace
  - Makefile.tau-mpi-pthread-pdt
  - Makefile.tau-multiplecounters-mpi-papi-pdt …
- Start with MPI instrumentation & PDT for automatic source instrumentation.

```bash
export TAU_MAKEFILE=<tau-dir>/bgI/lib/Makefile.tau-mpi-pdt
export TAU_MAKEFILE=<tau-dir>/bgp/lib/Makefile.tau-mpi-pdt
```
TAU Shell Scripts

- Compile your code with TAU shell scripts.

<table>
<thead>
<tr>
<th>GNU Compilers</th>
<th>IBM XL Compilers</th>
<th>TAU shell scripts</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpicc</td>
<td>mpixlc</td>
<td>mpixlc_r</td>
</tr>
<tr>
<td>mpicxx</td>
<td>mpixlcxx</td>
<td>mpixlcxx_r</td>
</tr>
<tr>
<td>mpiF77</td>
<td>mpixlf77</td>
<td>mpixlf77_r</td>
</tr>
<tr>
<td>mpiF90</td>
<td>mpixlf90</td>
<td>mpixlf90_r</td>
</tr>
</tbody>
</table>

- If your Fortran code is a fixed-format Fortran code, use “tau_f90.sh -qfixed”

- **-D options to XLF**: The XL Fortran compilers require a slightly different syntax to define preprocessor macro symbols. Instead of just "-D", you should use "-WF,-D" like this: mpiF77 -WF,-DMYFLAG myprogram.F
Analyze Performance Data

- **pprof** (for text based display)
  - sorts and displays profile data generated by TAU.
  - Execute pprof in the directory where profile files are located.

- **paraprof** (for GUI display)
  - TAU has Java based performance data viewer.
  - Requires Java1.4 or above, add it to your path.
  - --pack options pack the data into packed (.ppk) format and it does not launch the paraprof GUI.

```
paraprof --pack filename
```

- To launch the GUI

```
paraprof filename.ppk
```
pprof (Text based display)

<table>
<thead>
<tr>
<th>Time</th>
<th>Exclusive msec</th>
<th>Inclusive total msec</th>
<th>#Call</th>
<th>#Subs</th>
<th>Inclusive Name usec/call</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0</td>
<td>0.081</td>
<td>21:16.904</td>
<td>1</td>
<td>2</td>
<td>1276904738 int main(int, char **) C</td>
</tr>
<tr>
<td>100.0</td>
<td>0.083</td>
<td>21:16.904</td>
<td>1</td>
<td>1</td>
<td>1276904059 int main(int, char **, INIT_DATA *, INIT.Physics *) C</td>
</tr>
<tr>
<td>100.0</td>
<td>0.083</td>
<td>21:16.904</td>
<td>1</td>
<td>1</td>
<td>1276904859 int main(int, char **) C =&gt; 1276904859 int main(int, char **, INIT_DATA *, INIT.Physics *) C</td>
</tr>
<tr>
<td>100.0</td>
<td>0.125</td>
<td>21:16.903</td>
<td>1</td>
<td>3</td>
<td>1276909376 int domaint(int, char **, INIT_DATA *, INIT.Physics *) C =&gt; void perform_initialization(int, char **, INIT_DATA *, INIT.Physics *) C</td>
</tr>
<tr>
<td>100.0</td>
<td>0.125</td>
<td>21:16.903</td>
<td>1</td>
<td>3</td>
<td>1276909376 void perform_initialization(int, char **, INIT_DATA *, INIT.Physics *) C</td>
</tr>
<tr>
<td>100.0</td>
<td>0.071</td>
<td>21:16.903</td>
<td>1</td>
<td>1</td>
<td>1276903684 void g_init(int, char ***, INIT.DATA *, INIT.Physics ) S ** C</td>
</tr>
<tr>
<td>100.0</td>
<td>0.071</td>
<td>21:16.903</td>
<td>1</td>
<td>1</td>
<td>1276903684 void perform_initialization(int, char **, INIT.DATA *, INIT.Physics ) S ** C</td>
</tr>
<tr>
<td>100.0</td>
<td>0.127</td>
<td>21:16.903</td>
<td>1</td>
<td>3</td>
<td>1276903613 void d_init(int, char ***, INIT.DATA *, INIT.Physics ) S ** C</td>
</tr>
<tr>
<td>100.0</td>
<td>0.127</td>
<td>21:16.903</td>
<td>1</td>
<td>3</td>
<td>1276903613 void g_init(int, char ***, INIT.DATA *, INIT.Physics ) S ** C</td>
</tr>
<tr>
<td>99.9</td>
<td>0.571</td>
<td>21:16.174</td>
<td>1</td>
<td>20</td>
<td>1276714150 void d_init(int, char ***, INIT.DATA *, INIT.Physics ) S ** C</td>
</tr>
<tr>
<td>99.9</td>
<td>0.571</td>
<td>21:16.174</td>
<td>1</td>
<td>20</td>
<td>1276714150 void set_up_cauchy_data(INIT_DATA *, INIT.Physics *) C</td>
</tr>
<tr>
<td>99.2</td>
<td>0.211</td>
<td>21:06.620</td>
<td>1</td>
<td>4</td>
<td>1266620615 void set_states(INIT_DATA *, INIT.Physics *, INIT.Physics *, INIT.Physics *, INIT.Physics <em>, INIT.Physics</em>) C</td>
</tr>
<tr>
<td>99.2</td>
<td>0.211</td>
<td>21:06.620</td>
<td>1</td>
<td>4</td>
<td>1266620615 void set_up_cauchy_data(INIT_DATA *, INIT.Physics *) C =&gt; void init_states(INIT_DATA *, INIT.Physics *, CHART *, INIT.SOLN **, RESTART DATA *) C</td>
</tr>
<tr>
<td>99.2</td>
<td>0.134</td>
<td>21:06.481</td>
<td>1</td>
<td>6</td>
<td>1266481343 void clip_front_to_subdomain(Front *) C</td>
</tr>
<tr>
<td>99.2</td>
<td>0.134</td>
<td>21:06.481</td>
<td>1</td>
<td>6</td>
<td>1266481343 void clip_front_to_subdomain(Front *) C</td>
</tr>
<tr>
<td>99.2</td>
<td>0.115</td>
<td>21:06.480</td>
<td>69</td>
<td>276</td>
<td>18354796 bool scatter_front(Front *) C</td>
</tr>
<tr>
<td>99.2</td>
<td>0.115</td>
<td>21:06.480</td>
<td>69</td>
<td>276</td>
<td>18354796 bool scatter_front(Front *) C</td>
</tr>
<tr>
<td>....</td>
<td>....</td>
<td>....</td>
<td>....</td>
<td>....</td>
<td>....</td>
</tr>
</tbody>
</table>

--More--
-PROFILE(default)
Identify the routines that use the most time
Show Thread Statistics Text Window

<table>
<thead>
<tr>
<th>Metric: Time</th>
<th>Sorted By: Exclusive</th>
<th>Units: microseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Total Time</td>
<td>Exclusive</td>
<td>Inclusive</td>
</tr>
<tr>
<td>24.6</td>
<td>1.997E8</td>
<td>1.997E8</td>
</tr>
<tr>
<td>23.9</td>
<td>1.946E8</td>
<td>1.946E8</td>
</tr>
<tr>
<td>13.8</td>
<td>1.125E8</td>
<td>1.125E8</td>
</tr>
<tr>
<td>31.0</td>
<td>5.695E7</td>
<td>2.522E8</td>
</tr>
<tr>
<td>3.1</td>
<td>1.628E7</td>
<td>2.520E7</td>
</tr>
<tr>
<td>3.2</td>
<td>1.076E7</td>
<td>2.589E7</td>
</tr>
<tr>
<td>32.2</td>
<td>9.608E5</td>
<td>2.612E8</td>
</tr>
<tr>
<td>1.8</td>
<td>8.674E5</td>
<td>1.455E7</td>
</tr>
<tr>
<td>4.2</td>
<td>7.529E5</td>
<td>3.379E7</td>
</tr>
<tr>
<td>1.2</td>
<td>6.966E7</td>
<td>9.371E6</td>
</tr>
<tr>
<td>12.8</td>
<td>6.794E3</td>
<td>1.038E8</td>
</tr>
<tr>
<td>3.5</td>
<td>6.164E9</td>
<td>2.074E7</td>
</tr>
<tr>
<td>0.8</td>
<td>5.535E7</td>
<td>6.427E7</td>
</tr>
<tr>
<td>0.7</td>
<td>4.890E5</td>
<td>5.800E4</td>
</tr>
<tr>
<td>97.9</td>
<td>4.801E8</td>
<td>7.968E8</td>
</tr>
<tr>
<td>14.0</td>
<td>4.201E8</td>
<td>1.134E8</td>
</tr>
<tr>
<td>8.7</td>
<td>4.070E9</td>
<td>7.054E8</td>
</tr>
<tr>
<td>11.5</td>
<td>3.829E10</td>
<td>9.375E8</td>
</tr>
<tr>
<td>1.9</td>
<td>3.723E5</td>
<td>1.548E8</td>
</tr>
<tr>
<td>0.8</td>
<td>3.489E23</td>
<td>6.714E10</td>
</tr>
<tr>
<td>0.4</td>
<td>3.384E33</td>
<td>3.531E8</td>
</tr>
<tr>
<td>1.6</td>
<td>2.621E8</td>
<td>2.963E7</td>
</tr>
<tr>
<td>2.3</td>
<td>3.218E8</td>
<td>6.916E7</td>
</tr>
<tr>
<td>2.3</td>
<td>3.171E11</td>
<td>1.865E8</td>
</tr>
<tr>
<td>1.0</td>
<td>3.139E7</td>
<td>8.442E9</td>
</tr>
<tr>
<td>11.8</td>
<td>3.274E9</td>
<td>3.684E7</td>
</tr>
<tr>
<td>11.8</td>
<td>2.876E26</td>
<td>9.673E7</td>
</tr>
<tr>
<td>0.9</td>
<td>2.845E36</td>
<td>6.961E24</td>
</tr>
<tr>
<td>2.3</td>
<td>2.752E10</td>
<td>1.904E8</td>
</tr>
<tr>
<td>1.0</td>
<td>2.661E11</td>
<td>4.600E9</td>
</tr>
<tr>
<td>12.3</td>
<td>2.595E26</td>
<td>9.941E7</td>
</tr>
<tr>
<td>0.4</td>
<td>2.564E37</td>
<td>3.224E47</td>
</tr>
<tr>
<td>2.8</td>
<td>2.462E27</td>
<td>2.243E7</td>
</tr>
<tr>
<td>1.9</td>
<td>1.871E16</td>
<td>1.537E8</td>
</tr>
<tr>
<td>0.3</td>
<td>2.178E10</td>
<td>2.614E7</td>
</tr>
<tr>
<td>78.4</td>
<td>1.750E11</td>
<td>6.380E10</td>
</tr>
<tr>
<td>7.1</td>
<td>1.553E9</td>
<td>5.762E8</td>
</tr>
<tr>
<td>19.7</td>
<td>1.447E8</td>
<td>1.604E8</td>
</tr>
<tr>
<td>1.0</td>
<td>1.427E8</td>
<td>7.862E10</td>
</tr>
<tr>
<td>8.8</td>
<td>1.387E8</td>
<td>5.193E8</td>
</tr>
<tr>
<td>1.0</td>
<td>1.352E8</td>
<td>7.802E8</td>
</tr>
</tbody>
</table>

Show Thread Statistics Text Window
Function Data and Comparison Windows

Name: HELMEOs [helm_eos.f] [391.7]–[1380.9]
Metric Name: Time
Value: Exclusive
Units: microseconds

ParaProf: Comparison Window
Metric: Time
Value: Exclusive
Units: microseconds

HELMEOS

MPL_Barrier()

MPI_Allreduce()

HELMSSTATE
Generate a Flat Profile

- export PATH=/gpfs/home1/tulin/TAUL/tau-2.17.3/ppc64/bin:$PATH
- export TAU_MAKEFILE=/gpfs/home1/tulin/TAUL/tau2.17.3/bgl/lib/Makefile.tau-mpi-pdt
- make CC=tau_cc.sh CXX=tau_cxx.sh F90='tau_f90.sh -qfixed'
- vi tau_app.run
  
In your job script file, set the environment variable PROFILEDIR. Provide the full path to the directory where you want to store the profile files.

```bash
# @ arguments = -np 16 -env PROFILEDIR=<profile-dir> -exe ...
- llsubmit tau_app.run
- cd <profile-dir>
- paraprof --pack tau_app.ppk
- paraprof tau_app.ppk
```
-PROFILECALLPATH

- Generate call path profiles
- Shows the time spend in a routine when it is called by another routine in the calling path.
  - f1 => f2 shows the time spent in f2 when it is called by f1
paraprof → Windows → Threads → Call Path Relations
Generate Callpath / Callgraph Profile

- export PATH=/gpfs/home1/tulin/TAUL/tau-2.17.3/ppc64/bin:$PATH
- export TAU_MAKEFILE=/gpfs/home1/tulin/TAUL/tau2.17.3/bgl/lib/Makefile.tau-callpath-mpi-pdt
- make CC=tau_cc.sh CXX=tau_cxx.sh F90=’tau_f90.sh -qfixed’
- vi tau_app.run

In your job script file, set the environment variable PROFILEDIR. Provide the full path to the directory where you want to store the profile files.

```bash
# @ arguments = -np 16 -env PROFILEDIR=<profile-dir> -exe …
```

- llsubmit tau_app.run
- cd <profile-dir>
- paraprof --pack tau_app.ppk
- paraprof tau_app.ppk
  (Windows → Thread → Call Graph
   → Call Path Relations)
Blue Gene modern CPUs provide on-chip hardware performance counters that can record several events.

- The number of instructions issued
- The number of L1, L2 and L3 data and instruction cache misses, hits, access, read, write.
- Floating point operations executed.

TAU uses the Performance Data Standard and API (PAPI-Performance Application Programming Interface) to access these performance counters.

Set the COUNTER\texttt{x} environment variables.
**Performance Counters on BG/L microprocessor (PowerPC440)**

Test case 8: Available events and hardware information.

<table>
<thead>
<tr>
<th>Name</th>
<th>Derived</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAPI_L3_TCM</td>
<td>No</td>
<td>Level 3 cache misses (BGL_UPC_L3_CACHE_MISS_DATA_WILL_BE_REQED_DDR)</td>
</tr>
<tr>
<td>PAPI_L3_LDM</td>
<td>Yes</td>
<td>Level 3 load misses (BGL_UPC_L3_MSHNDLR_T0OK_REQ_RDQ, BGL_UPC_L3_MSHNDLR_T0OK_REQ_RDQ)</td>
</tr>
<tr>
<td>PAPI_L3_STM</td>
<td>No</td>
<td>Level 3 store misses (BGL_UPC_L3_MSHNDLR_T0OK_REQ_WBUF)</td>
</tr>
<tr>
<td>PAPI_FMA_INS</td>
<td>No</td>
<td>FMA instructions completed (BGL_FPU_ARITH_TRINARY_OP)</td>
</tr>
<tr>
<td>PAPI_TOT_CYC</td>
<td>No</td>
<td>Total cycles (Timebase register (null))</td>
</tr>
<tr>
<td>PAPI_L2_DCH</td>
<td>Yes</td>
<td>Level 2 data cache hits (BGL_UPC_PU0_PREF_STREAM_HIT, BGL_UPC_PU1_PREF_STREAM_HIT)</td>
</tr>
<tr>
<td>PAPI_L2_DCA</td>
<td>Yes</td>
<td>Level 2 data cache accesses (BGL_UPC_PU0_PREF_REQ_VALID, BGL_UPC_PU1_PREF_REQ_VALID)</td>
</tr>
<tr>
<td>PAPI_L3_TCH</td>
<td>No</td>
<td>Level 3 total cache hits (BGL_UPC_L3_CACHE_HIT)</td>
</tr>
<tr>
<td>PAPI_FLM_INS</td>
<td>No</td>
<td>Floating point multiply instructions (BGL_FPU_ARITH_MULTI_DIV)</td>
</tr>
<tr>
<td>PAPI_FAD_INS</td>
<td>No</td>
<td>Floating point add instructions (BGL_FPU_ARITH_ADD_SUBTRACT)</td>
</tr>
<tr>
<td>PAPI_BGL_QED</td>
<td>No</td>
<td>Floating point Oedipus operations (BGL_FPU_ARITH_OEDIPUS_OP)</td>
</tr>
<tr>
<td>PAPI_BGL_TS_32B</td>
<td>Yes</td>
<td>32B chunks sent in any torus link (BGL_UPC_TS_XM_32B_CHUNKS, BGL_UPC_TS_XP_32B_CHUNKS, BGL_UPC_TS_YM_32B_CHUNKS, BGL_UPC_TS_YP_32B_CHUNKS)</td>
</tr>
<tr>
<td>PAPI_BGL_TS_FULL</td>
<td>Yes</td>
<td>CLOCK^2 cycles with no torus token (accum) (BGL_UPC_TS_XM_LINK_AVLND_NO_VCD0_VCD_VCBN_TOKENS, BGL_UPC_TS_XP_LINK_AVLND_NO_VCD0_VCD_VCBN_TOKENS)</td>
</tr>
<tr>
<td>PAPI_BGL_TR_DPKTS_SENT</td>
<td>Yes</td>
<td>Data packets sent on any tree channel (BGL_UPC_TR_SNDR_2_VC1_DPKTS_SENT, BGL_UPC_TR_SNDR_2_VC1_DPKTS_SENT)</td>
</tr>
<tr>
<td>PAPI_BGL_TR_RCV_0_VC1_FULL</td>
<td>Yes</td>
<td>CLOCK^2 cycles with tree receiver full (accum) (BGL_UPC_TR_RCV_0_VC0_FULL, BGL_UPC_TR_RCV_1_VC0_FULL, BGL_UPC_TR_RCV_1_VC1_FULL)</td>
</tr>
</tbody>
</table>
Performance Counters on BG/P microprocessor (PowerPC450)

```
FAPI_L1_DCM_idx - 0, /*Level 1 data cache misses */
FAPI_L1_ICM_idx,  /*Level 1 instruction cache misses */
FAPI_L2_DCM_idx,  /*Level 2 data cache misses */
FAPI_L2_ICM_idx,  /*Level 2 instruction cache misses */
FAPI_L3_DCM_idx,  /*Level 3 data cache misses */
FAPI_L3_ICM_idx,  /*Level 3 instruction cache misses */
FAPI_L1_TCM_idx,  /*Level 1 total cache misses */
FAPI_L2_TCM_idx,  /*Level 2 total cache misses */
FAPI_L3_TCM_idx,  /*Level 3 total cache misses */
FAPI_CA_SNP_idx,  /*Snoops */
FAPI_CA_SHR_idx,  /*Request for shared cache line (SMP) */
FAPI_CA_CLN_idx,  /*Request for clean cache line (SMP) */
FAPI_CA_INV_idx,  /*Request for cache line invalidation (SMP) */
FAPI_CA_ITV_idx,  /*Request for cache line intervention (SMP) */
FAPI_L3_LDM_idx,  /*Level 3 load misses */
FAPI_L3_STM_idx,  /*Level 3 store misses */
/* 0x10 */
FAPI_BRU_IDL_idx, /*Cycles branch units are idle */
FAPI_FXU_IDL_idx, /*Cycles integer units are idle */
FAPI_FPU_IDL_idx, /*Cycles floating point units are idle */
FAPI_LSU_IDL_idx, /*Cycles load/store units are idle */
FAPI_TLB_DM_idx,  /*Data translation lookaside buffer misses */
FAPI_TLB_IM_idx,  /*Instruction translation lookaside buffer misses */
FAPI_TLB_TL_idx,  /*Total translation lookaside buffer misses */
FAPI_L1_LDM_idx,  /*Level 1 load misses */
FAPI_L1_STM_idx,  /*Level 1 store misses */
FAPI_L2_LDM_idx,  /*Level 2 load misses */
FAPI_L2_STM_idx,  /*Level 2 store misses */
FAPI_BTAC_M_idx,  /*BTAC miss */
FAPI_PRF_DM_idx,  /*Prefetch data instruction caused a miss */
FAPI_L3_DCH_idx,  /*Level 3 Data Cache Hit */
FAPI_TLB_SD_idx,  /*Klination lookaside buffer shootdowns (SMP) */
FAPI_CSFR_FAL_idx, /*Failed store conditional instructions */
```
Generate Hardware Counter Profile

- export PATH=/gpfs/home1/tulin/TAUP/tau-2.17.3/ppc64/bin:$PATH
- export TAU_MAKEFILE=/gpfs/home1/tulin/TAUP/tau2.17.3/bgp/lib/Makefile.tau-multiplecounters-mpi-pdt
- make CC=tau_cc.sh CXX=tau_cxx.sh F90='tau_f90.sh -qfixed'
- vi tau_app.run
  In your job script file, set the environment variable COUNTERx
  # @ arguments = -np 16 -env PROFILEDIR=<profile-dir>
    -env COUNTER1=GET_TIME_OF_DAY
    -env COUNTER2=PAPI_L1_DCM
    -env COUNTER3=PAPI_L1_DCH -exe ...
- llsubmit tau_app.run
- cd <profile-dir>
- paraprof --pack tau_app.ppk
- paraprof tau_app.ppk
Performance Counters
Fast Blue Gene Timers

- Blue Gene systems have a special clock cycle counter that can be used for low overhead timings,
  - BGLTIMERS: Use fast low-overhead timers on IBM BG/L
  - BGPTIMERS: Use fast low-overhead timers on IBM BG/P
  - LINUXTIMERS: Use low overhead TSC Counter for wallclock time.
  - CPUTIME: Use usertime+system time instead of wallclock time.
  - PAPIWALLCLOCK: Use PAPI to access wallclock time.
PerfExplorer

- Framework for parallel performance data mining.
- Enables the development and integration of data mining operations that will be applied to large-scale parallel performance profiles.
- Requires Java Run Time Environment 5
- Requires PerfDMF (Performance Data Management Framework) from TAU.
Relative Speedup

Speed Up \( S = \frac{t_s}{t_p} \)
Relative Efficiency – New Application:Time

Efficiency \( E = \frac{S}{p} \)
Using perfexplorer

- .ppk files are generated
  - llsubmit tau_app32.run
    - paraprof --pack tau_np32.ppk
  - ...
  - llsubmit tau_app512.run
    - paraprof --pack tau_np512.ppk
- Make sure you have Java5 or better in your PATH.
- paraprof
  - Add trial to the DB.
    - Trial type: Paraprof Packed Profile
    - Select File(s) -> OK.
    - Uploading Trial.
- Perfexplorer
  - Choose Experiments.
  - Charts → Relative SpeedUp
Part II: Performance Issues

How to program machines that can be built;
How to build machines that can be programmed

James Glimm

Chair, Dept Applied Mathematics and Statistics, Stony Brook University

Associate Director for Science, NYCCS
The crisis in computing

- Clock speeds have hit technology and economic limits
- Increased speed depends on increased parallelism
- Network cost (all to all) is $N^2$ (not scaling)
- Mesh network (BG/L) solved this problem but reached a limit
- Modern computers will have more cores per chip
- How to program?
Hybrid Architectures

- 8-32 or more cores per chip
- Shared memory (threads), one per core on chip; many chips
- MPI across nodes, OpenMP within nodes
- What is the programming model which will allow use of such hardware?
- Answer: three levels of mesh decomposition
- Many such proposals from the computer science community
Three mesh levels

- Finest (usual) mesh: for computation of PDE solutions
- Middle level (thread) mesh:
  - Each block belongs to one thread, and has a fraction of memory of chip for writing
- Coarsest level (MPI) mesh:
  - Each block belongs to one chip; messages between blocks. Each thread can read from entire memory of chip, but can write only to restricted memory of thread
Three Mesh Levels:
4X4 (MPI) X 8X8 (threads) X 8x8 (PDE computational cells)

Thread mesh block (8x8 of these per processor block). Allows 64 threads (cores) per processor.

Global read and local write across processor mesh block.

Processor Mesh block (4x4 of these)
MPI across boundaries

single computational cell for PDE solution. 8x8 of these across single thread mesh block.
MPI efficiency

- Depends on large MPI blocks for messages
- Messages only for border of MPI block, not for interior
- Depends on surface \(<<\) volume, i.e. requires big blocks
- Remains true for coarse (MPI) mesh, which governs the messages
Programming Model

- Smart MPI functions
- Between coarse (message) blocks: as usual
- Between mid-sized (thread) blocks: only copy or read
- To single mid-sized (thread) block: writing by only one thread allowed.
Proposal

- Move from computer science tests to science/engineering tests and codes
- Develop programming model for scientific codes and test on BG/P and on BG/Q when available
- Work with IBM scientists toward this goal
- Limited program modifications
  - Redefine MPI routines
  - Redefine ghost cell memory allocation at borders of coarse mesh blocks only
References

- http://www.cs.uoregon.edu/research/tau/home.php
- http://www.cs.uoregon.edu/research/tau/docs/newguide/index.html
- http://icl.cs.utk.edu/projects/papi
- http://acts.nersc.gov/
THANK YOU!