Preparing an application for Hybrid Supercomputing

Larry Kaplan
Senior Distinguished Technologist

John M Levesque
Chief Technologist for Cray Programming Environment
Senior Distinguished Technologist

John.levesque@hpc.com
Agenda

- Introduction and a Little History
- Steps in moving an application to a GPU
- A simple example - himeno
- Optimizing himeno – with openmp on the node
- Optimizing himeno – with openmp offload on the GPUs (or Openacc). DEMO
- Dealing with Halo exchanges DEMO
Introduction

Supercomputing and I have been around for around 55 years

The architecture we have today was first introduced in the last 55 years

We should learn from the past to prepare for the future

High level parallelism between many nodes – 1995 Beowulf

MIMD shared memory parallelism within the node – 1985 with Cray X-MP

Low level SIMD Parallelism – 1970 ILLIAC IV
Who is John Levesque?

- 1964 – 1968
  - Delivered Meads Fine Bread, (Student at University of New Mexico 62-72)
  - Sandia Laboratories – Underground Physics Department (CDC3600)
- 1970 – 1972
  - Air Force Weapons Laboratory (CDC 6600)
- 1972 – 1978
  - R & D Associates - (CDC 7600, Illiac IV, Cray 1, Cyber 203-205)
- 1978 -1979
  - Massachusetts Computer Associates (Iliac IV, Cray 1) Parallizer
- 1979 – 1991
  - Pacific Sierra Research – (Intel Paragon, Ncube, Univac APS) - VAST
  - Applied Parallel Research – (MPPs of all sorts, CM5, NEC SX) – FORGE
- 1998 – 2001
  - IBM Research – Director Advanced Computing Technology Center
  - Times N Systems – Director of Software Development
- 9/2001 – 1/2021
  - Cray Inc – Director – Cray’s Supercomputing Center of Excellence
- 1/2021 – 7/2024
  - HPE HPC/AI– Senior Distinguished Technologist – CTO Office
Complete and fully supported software development Suite

**APPLICATION DEVELOPMENT**

- C/C++ and Fortran Compilers
  *Deliver mature vectorizing & parallelizing technology*
- I/O, scientific & math libraries
  *Scientific libraries integrated with CCE*
- Cray MPI
  *Scalable communication across many nodes*
- Deep learning plug-in
  *Easily scale frameworks across many nodes*

**DEBUGGING**

- **Abnormal Termination Processing**
  *Manage core files at scale*
- **STAT**
  *Stack trace analysis at scale*

**PERFORMANCE ANALYSIS, PORTING, AND OPTIMIZATION**

- **Comparative Debugger**
  *Compare two versions of an application*
- **GDB for HPC**
  *Parallel gdb for scalable debugging*
- **Valgrind for HPC**
  *Memory debugging at scale*

- **Performance Analysis Tools**
  *Simple and advanced interfaces provide whole program profiling + visualization*
- **Code Parallelization Assistant**
  *Reveal hidden potential of an application via code restructuring*
Providing the user with compiler choice

- Use modules to select compiling environment
- Automatically uses our math, scientific, and communication libraries with chosen compiler
- Can use debug and profiling tools with chosen compiler

<table>
<thead>
<tr>
<th>Compiler Environment</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPE Cray Programming Environment (CCE)</td>
<td>Cray MPI and SHMEM</td>
</tr>
<tr>
<td>AMD Programming Environment</td>
<td>Performance Analysis Tools</td>
</tr>
<tr>
<td>Intel Programming Environment</td>
<td>Debugger Support Tools</td>
</tr>
<tr>
<td>NVIDIA Programming Environment</td>
<td>Scientific and Math Libraries</td>
</tr>
<tr>
<td>GNU Programming Environment</td>
<td>Environment Setup and Compiling Support</td>
</tr>
<tr>
<td></td>
<td>Third Party Products</td>
</tr>
</tbody>
</table>
Performance Analysis Tools
Reduce time and effort associated with porting and tuning of applications on HPE systems

Highlights:

- Different tools to fit different developer needs—from quick visual analysis to variety of different experiments, integration with compilers and more...
- Target **scalability** issues in all areas of tool development—designed to improve performance on the largest of systems
- Provide **whole program performance analysis** across many nodes to identify critical performance bottlenecks in a program
- Help to uncover issues but also **suggestions to improve performance**
- Unique and valuable **load imbalance analysis**
- Target **ease of use** with simple and advanced user interfaces
- Supports **programs** written in Fortran, C or C++ with MPI, SHMEM, UPC, OpenMP or OpenACC, CUDA or HIP, and their combinations.

Our performance tools profiled production applications with over 256,000 ranks.
Code Parallelization Assistant

- Reduce effort associated with adding OpenMP to MPI programs
- Works in conjunction with our compiler and performance tools
- Easily navigate through source code to highlighted dependences or bottlenecks
- Identify work-intensive loops to parallelize, perform dependence analysis, scope variables and generate OpenMP directives
- Great first step when moving large, complex loops to GPUs
**Directive-Based Programming Models**

- Huge potential to provide cross-architecture portability (CPUs and GPUs)
- Standard specifications that all compiler vendors can implement
- Performance portability across vendors has been a recent challenge
- Has been critical for Fortran, especially for offloading
- OpenMP and OpenACC
  - Continued participation in language committees
  - Ongoing support for current and future specifications
  - Leverage common compiler and library codebase for OpenMP and OpenACC implementations
  - Significant opportunities for general performance improvements
  - Significant opportunities for improving construct-to-hardware mapping
    - Better cross-vendor consistency
    - Better use of descriptive features (e.g., “omp loop”)
    - Use of multidimensional grids, especially for “collapse” loops
- Specialized, low-level directive-based models may expand user base
  - Optimize performance for a limited set of OpenMP/OpenACC constructs and APIs
  - Provide a portable model similar to existing kernel languages (e.g., CUDA or HIP)
Using *perftools-lite* (or *-loops* or *-hbm*)

- module load *perftools-lite* or *perftools-lite-loops* or *perftools-lite-hbm*
  - Module *perftools-base* should already be loaded

- Build application

- Run application

- Statistics report comes out within standard out
  - Also generates a directory of profile data to be examined with different options
Let's start with a simple example

- cd himeno_hybrid
- module load perftools-lite
- ftn -rm himenoBMTxpr.f -ohimeno

[levesque@o185i082 himeno_hybrid]$ ftn -rm himenoBMTxpr.f -ohimeno
WARNING: PerfTools is saving object files from a temporary directory into directory '/lvol/levesque/.craypat/himenox/507153'
INFO: creating the PerfTools-instrumented executable 'himenox' (lite-samples) ...OK
- srun -n 2 ./himenox
Perftools-lite profile – Run on 1 nodes–2 MPI tasks/node

Table 1: Profile by Function

<table>
<thead>
<tr>
<th>Samp%</th>
<th>Samp</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Function=[MAX10]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PE=HIDE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Samp</th>
<th>Samp%</th>
<th>Imb.</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td>6,421.0</td>
<td>--</td>
<td>--</td>
<td>Total</td>
</tr>
<tr>
<td>99.0%</td>
<td>6,359.0</td>
<td>--</td>
<td>--</td>
<td>USER</td>
</tr>
<tr>
<td>98.8%</td>
<td>6,341.0</td>
<td>51.0</td>
<td>1.6%</td>
<td>jacobi_</td>
</tr>
</tbody>
</table>

Exclusive time
Sampling is in 100\(^{th}\) of a second
Imbalance
Only showing items that take up more than 1% of time – you can override with -T
**Perftools-lite profile – Run on 1 nodes–2 MPI tasks/node**

Table 1: Profile by Function

<table>
<thead>
<tr>
<th>Samp%</th>
<th>Samp</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PE=HIDE</td>
</tr>
<tr>
<td>100.0%</td>
<td>6,421.0</td>
<td>--</td>
<td>--</td>
<td>Total</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>USER</th>
</tr>
</thead>
<tbody>
<tr>
<td>99.0%</td>
<td>6,359.0</td>
<td>--</td>
<td>--</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>jacobi_</th>
</tr>
</thead>
<tbody>
<tr>
<td>98.8%</td>
<td>6,341.0</td>
<td>51.0</td>
<td>1.6%</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>initmt_</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3%</td>
<td>18.0</td>
<td>0.0</td>
<td>0.0%</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.7%</td>
<td>47.5</td>
<td>--</td>
<td>--</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>MPI_WAITALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.7%</td>
<td>46.5</td>
<td>45.5</td>
<td>98.9%</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>MPI_ISEND</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0%</td>
<td>0.5</td>
<td>0.5</td>
<td>100.0%</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>mpi_isend_</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0%</td>
<td>0.5</td>
<td>0.5</td>
<td>100.0%</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>ETC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2%</td>
<td>14.5</td>
<td>4.5</td>
<td>47.4%</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>__cray_memcpy_ROME</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2%</td>
<td>14.5</td>
<td>4.5</td>
<td>47.4%</td>
<td></td>
</tr>
</tbody>
</table>

*Exclusive time
Sampling is in 100th of a second
Imbalance*
Table 2: Profile of maximum function times

<table>
<thead>
<tr>
<th>Samp%</th>
<th>Samp</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td>6,392.0</td>
<td>51.0</td>
<td>1.6%</td>
<td>jacobi</td>
</tr>
<tr>
<td>100.0%</td>
<td>6,392.0</td>
<td>--</td>
<td>--</td>
<td>pe.0</td>
</tr>
<tr>
<td>98.4%</td>
<td>6,290.0</td>
<td>--</td>
<td>--</td>
<td>pe.1</td>
</tr>
<tr>
<td>1.4%</td>
<td>92.0</td>
<td>45.5</td>
<td>98.9%</td>
<td>MPI_WAITALL</td>
</tr>
<tr>
<td>1.4%</td>
<td>92.0</td>
<td>--</td>
<td>--</td>
<td>pe.1</td>
</tr>
<tr>
<td>0.0%</td>
<td>1.0</td>
<td>--</td>
<td>--</td>
<td>pe.0</td>
</tr>
<tr>
<td>0.3%</td>
<td>19.0</td>
<td>4.5</td>
<td>47.4%</td>
<td>__cray_memcpy_ROME</td>
</tr>
<tr>
<td>0.3%</td>
<td>19.0</td>
<td>--</td>
<td>--</td>
<td>pe.1</td>
</tr>
<tr>
<td>0.2%</td>
<td>10.0</td>
<td>--</td>
<td>--</td>
<td>pe.0</td>
</tr>
<tr>
<td>0.3%</td>
<td>18.0</td>
<td>0.0</td>
<td>0.0%</td>
<td>initmt</td>
</tr>
<tr>
<td>0.3%</td>
<td>18.0</td>
<td>--</td>
<td>--</td>
<td>pe.0</td>
</tr>
<tr>
<td>0.3%</td>
<td>18.0</td>
<td>--</td>
<td>--</td>
<td>pe.1</td>
</tr>
</tbody>
</table>
Perftools-lite profile – Run on 1 nodes–2 MPI tasks/node

Table 4: Program HW Performance Counter Data

pat_report -T himenox+507859-4242835s>profile_T

<table>
<thead>
<tr>
<th>Category</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread Time</td>
<td>64.264144 secs</td>
</tr>
<tr>
<td>CORE_TO_L2_CACHEABLE_REQUEST_ACCESS_STATUS:</td>
<td></td>
</tr>
<tr>
<td>LS_RD_BLK_C</td>
<td>5,977,290,738</td>
</tr>
<tr>
<td>L2_PREFETCH_HIT_L2</td>
<td>1,168,069,330</td>
</tr>
<tr>
<td>L2_PREFETCH_HIT_L3</td>
<td>2,141,277,288</td>
</tr>
<tr>
<td>REQUESTS_TO_L2_GROUP1:L2_HW_PF</td>
<td>3,775,988,456</td>
</tr>
<tr>
<td>REQUESTS_TO_L2_GROUP1:RD_BLK_X</td>
<td>513,939,922</td>
</tr>
<tr>
<td>Cache Lines PF from OffCore</td>
<td>0.041G/sec 2,607,919,126 lines</td>
</tr>
<tr>
<td>Cache Lines PF from Memory</td>
<td>0.007G/sec 466,641,838 lines</td>
</tr>
<tr>
<td>Cache Lines Requested from Memory</td>
<td>0.017G/sec 1,069,532,376 lines</td>
</tr>
<tr>
<td>Write Memory Traffic GBytes</td>
<td>0.063G/sec 4.06 GB</td>
</tr>
<tr>
<td>Read Memory Traffic GBytes</td>
<td>1.530G/sec 98.32 GB</td>
</tr>
<tr>
<td>Memory traffic GBytes</td>
<td>1.593G/sec 102.38 GB</td>
</tr>
<tr>
<td>Memory Traffic / Nominal Peak</td>
<td>0.8%</td>
</tr>
</tbody>
</table>
Perftools-lite profile – Run on 1 nodes–2 MPI tasks/node

Table 4: Program HW Performance Counter Data

| Avg Write Time per Writer Rank | Avg Write Time per Writer Rank | Write Rate MiBytes/sec | Number of Writes Writer per Ranks | Avg Bytes/Call PE=HIDE
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>453.000020</td>
<td>0.0000008</td>
<td>0.433920</td>
<td>2</td>
<td>1.0</td>
</tr>
<tr>
<td>0.0000003</td>
<td>0.0000356</td>
<td>108.583797</td>
<td>2</td>
<td>11.0</td>
</tr>
</tbody>
</table>

Notes for table 5:

This table shows the average time and number of bytes written to each output file, taking the average over the number of ranks that wrote to the file. It also shows the number of write operations, and average rates.

For further explanation, see the "General table notes" below, or use: pat_report -v -O write_stats ...

Table 5: File Output Stats by Filename

<table>
<thead>
<tr>
<th>Avg Write Time per Writer Rank</th>
<th>Avg Write Time per Writer Rank</th>
<th>Write Rate MiBytes/sec</th>
<th>Number of Writes Writer per Ranks</th>
<th>Avg Bytes/Call PE=HIDE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000018</td>
<td>0.000008</td>
<td>0.433920</td>
<td>2</td>
<td>1.0</td>
</tr>
<tr>
<td>0.000003</td>
<td>0.000036</td>
<td>108.583797</td>
<td>2</td>
<td>11.0</td>
</tr>
</tbody>
</table>
**Perftools-lite profile – Run on 1 nodes–2 MPI tasks/node**

Table 3: Profile by Group, Function, and Line

```
<table>
<thead>
<tr>
<th>Samp%</th>
<th>Samp</th>
<th>Imb.</th>
<th>Samp</th>
<th>Imb.</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0%</td>
<td>6,421.0</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>Total</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>------</td>
<td>-------</td>
<td>------</td>
<td>-----------</td>
</tr>
<tr>
<td>99.0%</td>
<td>6,359.0</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>USER</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>------</td>
<td>-------</td>
<td>------</td>
<td>-----------</td>
</tr>
<tr>
<td>98.8%</td>
<td>6,341.0</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>jacobi_3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>/vol/levesque/himeno_hybrid/himenoBMTxpr.f</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>------</td>
<td>-------</td>
<td>------</td>
<td>-----------</td>
</tr>
<tr>
<td>4</td>
<td>1.6%</td>
<td>105.5</td>
<td>1.5</td>
<td>2.8%</td>
<td>line.89</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>------</td>
<td>-------</td>
<td>------</td>
<td>-----------</td>
</tr>
<tr>
<td>4</td>
<td>0.0%</td>
<td>1.0</td>
<td>1.0</td>
<td>100.0%</td>
<td>line.213</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>------</td>
<td>-------</td>
<td>------</td>
<td>-----------</td>
</tr>
<tr>
<td>4</td>
<td>85.9%</td>
<td>5,518.5</td>
<td>6.5</td>
<td>0.2%</td>
<td>line.214</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>------</td>
<td>-------</td>
<td>------</td>
<td>-----------</td>
</tr>
<tr>
<td>4</td>
<td>5.5%</td>
<td>352.0</td>
<td>18.0</td>
<td>9.7%</td>
<td>line.224</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>------</td>
<td>-------</td>
<td>------</td>
<td>-----------</td>
</tr>
<tr>
<td>4</td>
<td>1.6%</td>
<td>101.0</td>
<td>8.0</td>
<td>14.7%</td>
<td>line.225</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>------</td>
<td>-------</td>
<td>------</td>
<td>-----------</td>
</tr>
<tr>
<td>4</td>
<td>3.5%</td>
<td>226.0</td>
<td>17.0</td>
<td>14.0%</td>
<td>line.226</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>------</td>
<td>-------</td>
<td>------</td>
<td>-----------</td>
</tr>
<tr>
<td>4</td>
<td>0.6%</td>
<td>37.0</td>
<td>1.0</td>
<td>5.3%</td>
<td>line.227</td>
</tr>
</tbody>
</table>
```

pat_report -T himenox+507859-4242835s>profile_T
This is why we used `rm` on `ftn` line (`-h list=a`)

```
C******************************************************************************
subroutine jacobi(nn,gosa)
C******************************************************************************
IMPLICIT REAL*4(a-h,o-z)

include 'mpif.h'
include 'param.h'

DO loop=1,nn
  gosa=0.0
  wgosa=0.0
  DO K=2,kmax-1
    DO J=2,jmax-1
      DO I=2,imax-1
        S0=a(I,J,K,1)*p(I+1,J,K)+a(I,J,K,2)*p(I,J+1,K)
        +a(I,J,K,3)*p(I,J,K+1)
        +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K))
        -p(I-1,J+1,K)+p(I-1,J-1,K))
        +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1))
        -p(I,J+1,K-1)+p(I,J-1,K-1))
        +c(I,J,K,1)*p(I-1,J,K)+c(I,J,K,2)*p(I,J-1,K)
        +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
        SS=(S0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
        WGOSA=WGOSA+SS*SS
        wrk2(I,J,K)=p(I,J,K)+OMEGA*SS
      enddo
    enddo
  enddo
enddo
C******************************************************************************
DO K=2,kmax-1
  DO J=2,jmax-1
    DO I=2,imax-1
      p(I,J,K)=wrk2(I,J,K)
    enddo
  enddo
enddo
```
## Perftools-lite profile – Run on 1 nodes–2 MPI tasks/node

### Table 3: Profile by Group, Function, and Line

<table>
<thead>
<tr>
<th>Samp%</th>
<th>Samp</th>
<th>Imb.</th>
<th>Samp%</th>
<th>Imb.</th>
<th>Group</th>
<th>Function</th>
<th>Source</th>
<th>Line</th>
<th>PE=HIDE</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td>6,421.0</td>
<td>--</td>
<td>--</td>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>99.0%</td>
<td>6,359.0</td>
<td>--</td>
<td>--</td>
<td>USER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>98.8%</td>
<td>6,341.0</td>
<td>--</td>
<td>--</td>
<td>jacobi</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>85.9%</td>
<td>5,518.5</td>
<td>6.5</td>
<td>0.2%</td>
<td>line.214</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.5%</td>
<td>352.0</td>
<td>18.0</td>
<td>9.7%</td>
<td>line.224</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.6%</td>
<td>101.0</td>
<td>8.0</td>
<td>14.7%</td>
<td>line.225</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.5%</td>
<td>226.0</td>
<td>17.0</td>
<td>14.0%</td>
<td>line.226</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.6%</td>
<td>37.0</td>
<td>1.0</td>
<td>5.3%</td>
<td>line.227</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
pat_report -T himenox+507859-4242835s>profile_T
```
Lets Optimize this code

- Shared memory parallelism
- Put on the GPU
- To do these things we need to understand the loop limits

module swap perftools-lite perftools-lite-loops
ftn -rm himenoBMTxpr.f -ohimenox
WARNING: PerfTools is saving object files from a temporary directory into directory
'/lvol/levesque/.craypat/himenox/508047'
INFO: creating the PerfTools-instrumented executable 'himenox' (lite-loops) ...OK
srun -n 2 ./himenox
Table 1: Calltree with Loop Inclusive Time

| Incl | Incl | Loop Exec | Loop | Calltree | Trips | PE-HIDE
| Time% | Time | Avg | | | |
| 100.0% | 63.50 | -- | -- | Total |

```
| 100.0% | 63.50 | -- | -- | himenobmtxp_ |
| 197.3% | 125.31 | 2,331,148 | 126.5 | jacobi_ |
| 3 | 99.4% | 63.14 | 2 | 74.0 | jacobi_.LOOP.1.li.208 |
| 4 | 98.6% | 62.61 | 148 | 125.0 | jacobi_.LOOP.2.li.211 |
| 4 | 0.3% | 0.21 | 18,500 | 125.0 | jacobi_.LOOP.6.li.232 |
| 5 | 0.2% | 0.10 | 2,312,500 | 126.5 | jacobi_.LOOP.7.li.233 |
| 3 | 0.6% | 0.35 | 16,256 | 128.5 | initmt_ |
| 3 | 0.3% | 0.20 | 1 | 128.0 | initmt_.LOOP.1.li.154 |
| 4 | 0.3% | 0.20 | 128 | 128.0 | initmt_.LOOP.2.li.155 |
| 5 | 0.3% | 0.19 | 16,384 | 128.0 | initmt_.LOOP.3.li.156 |
| 3 | 0.2% | 0.16 | 1 | 127.0 | initmt_.LOOP.4.li.175 |
| 3 | 0.0% | 0.00 | 0 | -- | _STOP3 |
| 3 | 0.0% | 0.00 | -- | -- | initcomm |
| 3 | 0.0% | 0.00 | 0 | -- | mpi_init_.(sync) |
| 3 | 0.0% | 0.00 | 0 | -- | MPI_INIT |
| 3 | 0.0% | 0.00 | 0 | -- | mpi_finalize_.(sync) |
| 3 | 0.0% | 0.00 | -- | -- | initmax_ |
| 3 | 0.0% | 0.00 | 1 | 2.0 | initmax_.LOOP.1.li.352 |
| 3 | 0.0% | 0.00 | 1 | 2.0 | initmax_.LOOP.2.li.359 |
```

This is why we used `–rm` on `ftn` line (`-h list=a`).

```fortran
C*******************************************************************************
200. subroutine jacobi(nn,gosa)
201. C*******************************************************************************
202. IMPLICIT REAL*4(a-h,o-z)
203. C
204. include 'mpif.h'
205. include 'param.h'
206. C
207. DO loop=1,nn
208. gosa=0.0
209. wgosa=0.0
210. DO K=2,kmax-1
211. DO J=2,jmax-1
212. DO I=2,imax-1
213. Vr3=0
214. S0=a(I,J,K,1)*p(I+1,J,K)+a(I,J,K,2)*p(I,J+1,K)
215. +a(I,J,K,3)*p(I,J,K+1)
216. +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K))
217. +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1))
218. +c(I,J,K,1)*p(I-1,J,K)+c(I,J,K,2)*p(I,J-1,K)
219. +c(I,J,K,3)*p(I,J,K-1)
220. SS=(S0*a(I,J,K,4)-p(I,J,K))*bnd
221. WGOSA=WGOSA+SS*SS
222. wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
223. C
224. DO K=2,kmax-1
225. DO J=2,jmax-1
226. DO I=2,imax-1
227. p(I,J,K)=wrk2(I,J,K)
228. enddo
229. enddo
230. enddo
231. DO K=2,kmax-1
232. DO J=2,jmax-1
233. DO I=2,imax-1
234. p(I,J,K)=wrk2(I,J,K)
235. enddo
236. enddo
237. enddo
```

Hewlett Packard Enterprise
Let's Optimize this code

- Shared memory parallelism
- We have a tool called reveal to help parallelize code for host and CPU
- We need to build a program library to use reveal
  
  module unload perftools-lite-loops
  
  ftn -hpl=himeno.pl himenoBMTxpr.f
  
  reveal himeno.pl himeno+x+508124-4242835t
Nested Loop View

<table>
<thead>
<tr>
<th>Loop Name</th>
<th>Inclusive Time (%)</th>
<th>Inclusive Time (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>jacobi().LOOP1.li.208</td>
<td>99.40</td>
<td>63.14</td>
</tr>
<tr>
<td>jacobi().LOOP2.li.211</td>
<td>98.60</td>
<td>62.61</td>
</tr>
</tbody>
</table>

Source: /vol/levesque/himeno_hybrid/himenoBMTxpr.f

DO loop=1,nn

gosa=0.0

wgosa=0.0

DO K=2,kmax-1

DO J=2,jmax-1

Scope Loops

List of Loops to be Scoped

Scope? Line # File or Source Line

- /vol/levesque/himeno_hybrid/himenoBMTxpr.f
- 211

DO K=2,kmax-1

Scope For CPU Scope For GPU Cancel Close
A loop starting at line 211 was not vectorized because a better candidate was found at line 220.
include 'mpiT.h'
include 'param.h'

L 208 DO loop=1,nn
209  gosa=0.0
210  wgosa=0.0
211
212
213
214 S0=a(I,J,K,1)*p(I+1,J,K)+a(I,J,K,2)*p(I,J+1,K
215  +a(I,J,K,3)*p(I,J,K+1)
216  +b(I,J,K,1)*(p(I+1,J,1)+p(I+1,J-1,K)
217  -p(I-1,J+1,K)+p(I-1,J-1,K))
218  +b(I,J,K,2)*(p(I+1,J,K+1)+p(I,J+1,K-1)
219  -p(I,J+1,J-1)+p(I,J-1,J-1,K))
220  +b(I,J,K,3)*(p(I,J+1,K+1)+p(I,J,K-1)
221  -p(I,J-1,K+1)+p(I,J-1,K-1))
222  +c(I,J,K,1)*p(I-1,J,K)+c(I,J,K,2)*p(I,J-1,K)
223  +c(I,J,K,3)*p(I-1,J+1,K)+wrk1(I,J,K)
224 SS=(S0*a(I,J,K,4)-p(I,J,K))*bndl(I,J,K)
225 WGOSA=WGOSA+SS*SS
226 wrk2(I,J,K)=p(I,J,K)+OMEGA *SS

Info - Line 208
- "mpi_allreduce" was not inlined because the compiler was unable to locate the routine.
- "sendp"(/vol/levesque/himeno_hybrid/himenoBMTxpr.f446) not flattened because "mpi_wait"/sendp_clone 508738 1657189072_3"(/vol/levesque/himeno_hybrid/himenoBMTxpr.f446) w
Now let's put it on the GPU

- Go back to Reveal and generate GPU code
A loop starting at line 211 was scoped without errors.
A loop starting at line 211 is flat (contains no external calls).
A loop starting at line 211 was not vectorized because a better candidate was found at line 213.
Directive inserted by Cray Reveal. May be incomplete.
SOMP target teams distribute
SOMP private(i, j, k, s0, ss)
SOMP& firstprivateimax, jmax, kmax, omega
SOMP& reduction(+: wgosa)
SOMP& map(tofrom: wgosa)
SOMP& map(always, to: a(:, :, :], b(:, :, :], bnd(:, :, :], c(:, :, :),
SOMP& p(:, :, :], wrk1(:, :, :))
SOMP& map(always, tofrom: wrk2(:, :, :))

DO K=2,kmax-1
  DO J=2,jmax-1
    DO I=2,imax-1
      S0=a(I,J,K,1)*p(I+1,J,K)+a(I,J,K,2)*p(I,J+1,K)
      +a(I,J,K,3)*p(I,J,K+1)
      +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K)
      -p(I-1,J+1,K)+p(I-1,J-1,K))
      +b(I,J,K,2)*(p(I+1,J+1,K+1)-p(I,J-1,K+1)
      -p(I,J+1,K-1)+p(I,J+1,K-1))
      +b(I,J,K,3)*(p(I+1,J+1,K+1)-p(I-1,J-1,K+1)
      -p(I+1,J+1,K-1)+p(I-1,J-1,K-1))
      +c(I,J,K,1)*p(I-1,J,K)+c(I,J,K,2)*p(I,J-1,K)
      +c(I,J,K,3)*p(I,J,K+1)+wrk1(I,J,K)
      SS=(S0*a(I,J,K,4)-p(I,J,K)+bnd(I,J,K)
      +WGOSA=WGOSA+SS*SS
      wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
    enddo
  enddo
enddo
Commands for using perftools

- module swap perftools-lite-loops perftools
- ftn -hlist=a -homp himenoBMTxpr.f -o himeno
- pat_build -u -g mpi himeno
- srun -n 2 /himeno+pat
- pat_report -T <directory generated by execution>
Table 1: Profile by Function Group and Function

<table>
<thead>
<tr>
<th>Time%</th>
<th>Time</th>
<th>Imb. Time</th>
<th>Imb.</th>
<th>Calls</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Time%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0%</td>
<td>517.005427</td>
<td>--</td>
<td>--</td>
<td>170,510.0</td>
<td>Total</td>
</tr>
<tr>
<td>93.2%</td>
<td>482.005140</td>
<td>--</td>
<td>--</td>
<td>40,012.0</td>
<td>OACC</td>
</tr>
<tr>
<td>86.8%</td>
<td>448.955431</td>
<td>20.973100</td>
<td>6.0%</td>
<td>10,000.0</td>
<td>jacobi_.ACC_COPY@li.213</td>
</tr>
<tr>
<td>5.5%</td>
<td>28.517275</td>
<td>2.813567</td>
<td>12.0%</td>
<td>10,000.0</td>
<td>jacobi_.ACC_COPY@li.239</td>
</tr>
<tr>
<td>0.5%</td>
<td>2.609518</td>
<td>0.576732</td>
<td>24.1%</td>
<td>10,000.0</td>
<td>jacobi_.ACC_SYNC_WAIT@li.213</td>
</tr>
<tr>
<td>0.2%</td>
<td>1.233697</td>
<td>0.0009083</td>
<td>1.0%</td>
<td>6.0</td>
<td>jacobi_.ACC_COPY@li.89</td>
</tr>
<tr>
<td>0.1%</td>
<td>0.687733</td>
<td>0.134373</td>
<td>21.8%</td>
<td>10,000.0</td>
<td>jacobi_.ACC_ASYNC_KERNEL@li.213</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000823</td>
<td>0.000119</td>
<td>16.8%</td>
<td>3.0</td>
<td>jacobi_.ACC_ASYNC_KERNEL@li.89</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000663</td>
<td>0.000142</td>
<td>23.4%</td>
<td>3.0</td>
<td>jacobi_.ACC_SYNC_WAIT@li.89</td>
</tr>
<tr>
<td>3.9%</td>
<td>20.382070</td>
<td>--</td>
<td>--</td>
<td>110,053.0</td>
<td>MPI</td>
</tr>
<tr>
<td>3.3%</td>
<td>17.007617</td>
<td>13.779150</td>
<td>59.7%</td>
<td>20,006.0</td>
<td>MPI_WAITALL</td>
</tr>
<tr>
<td>0.5%</td>
<td>2.339848</td>
<td>1.328698</td>
<td>48.3%</td>
<td>40,012.0</td>
<td>MPI_ISEND</td>
</tr>
<tr>
<td>0.2%</td>
<td>0.984848</td>
<td>0.743195</td>
<td>57.3%</td>
<td>40,012.0</td>
<td>MPI_Irecv</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.049582</td>
<td>0.0002511</td>
<td>6.4%</td>
<td>10,004.0</td>
<td>MPI_ALLREDUCE</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000066</td>
<td>0.0000003</td>
<td>5.3%</td>
<td>1.0</td>
<td>MPI_CART_CREATE</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000032</td>
<td>0.000005</td>
<td>17.7%</td>
<td>1.0</td>
<td>MPI_Reduce</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000027</td>
<td>0.000004</td>
<td>16.0%</td>
<td>2.0</td>
<td>MPI_BARRIER</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000021</td>
<td>0.000002</td>
<td>10.9%</td>
<td>4.0</td>
<td>MPI_WTIME</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000008</td>
<td>0.000006</td>
<td>52.7%</td>
<td>2.0</td>
<td>MPI_TYPE_COMMIT</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000008</td>
<td>0.000001</td>
<td>20.2%</td>
<td>2.0</td>
<td>MPI_TYPE_VECTOR</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000005</td>
<td>0.000000</td>
<td>3.9%</td>
<td>1.0</td>
<td>MPI_CART_GET</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000003</td>
<td>0.000000</td>
<td>0.7%</td>
<td>2.0</td>
<td>MPI_CART_SHIFT</td>
</tr>
</tbody>
</table>
```c
205.  C
206.      include 'mpif.h'
207.      include 'param.h'
208.  C
209.  + 1-------------- DO loop=1,nn
210.  1      gosa=0.0
211.  1      wgosa=0.0
212.  1      ! Directive inserted by Cray Reveal. May be incomplete.
213.  + 1 MG<!OMP target teams distribute
214.  1 MG<!OMP& private( i, j, k, s0, ss)
215.  1 MG<!OMP& firstprivate( imax, jmax, kmax, omega)
216.  1 MG<!OMP& map(tofrom: wgosa)
217.  1 MG<!OMP& reduction(+: wgosa)
218.  1 MG<!OMP& map(always, to: a(i,j,k), b(i,j,k), bnd(i,j,k), c(i,j,k))
219.  1 MG<!OMP& p(i,j,k), wrk1(i,j,k)
220.  1 MG<!OMP& map(always, tofrom: wrk2(i,j,k))
221.  1 MG g-------- DO K=2,kmax-1
222.  + 1 MG g 4------ DO J=2,jmax-1
223.  1 MG g 4 gr3-- DO I=2,imax-1
224.  1 MG g 4 gr3  S0=a(i,j,k,1)*p(i+1,j,k)+a(i,j,k,2)*p(i,j+1,k)
225.  1 MG g 4 gr3  +a(i,j,k,3)*p(i,j,k+1)
226.  1 MG g 4 gr3  +b(i,j,k,1)*(p(i+1,j+1,k)-p(i+1,j-1,k))
227.  1 MG g 4 gr3  -p(i-1,j+1,k)+p(i-1,j-1,k))
228.  1 MG g 4 gr3  +b(i,j,k,2)*(p(i,j+1,k+1)-p(i,j-1,k+1))
229.  1 MG g 4 gr3  -p(i,j+1,k-1)+p(i,j-1,k-1))
230.  1 MG g 4 gr3  +b(i,j,k,3)*(p(i+1,j,k)-p(i-1,j,k))
231.  1 MG g 4 gr3  -p(i+1,j,k-1)+p(i-1,j,k-1))
232.  1 MG g 4 gr3  +c(i,j,k,1)*p(i-1,j,k)+c(i,j,k,2)*p(i,j-1,k)
233.  1 MG g 4 gr3  +c(i,j,k,3)*p(i,j,k+1)+wrk1(i,j,k)
234.  1 MG g 4 gr3  SS=(S0*a(i,j,k,4)-p(i,j,k))*bnd(i,j,k)
235.  1 MG g 4 gr3  WGOsa=wgosa+SS*SS
236.  1 MG g 4 gr3  WRK1(i,j,k)=WGOsa+WGOsa*SS
237.  1 MG g 4 gr3  enddo
238.  1 MG g 4 gr3  enddo
239.  1 MG g---------- enddo
```
DO K=2,kmax
   DO J=2,jmax
      DO I=2,imax
         p(I,J,K)=wrk2(I,J,K)
      enddo
   enddo
enddo

CALL sendp(ndx,ndy,ndz) ! P halo exchange
CALL mpi_allreduce(wgosA,gosa,1,mpi_real4,mpi_sum,mpi_comm_world,ierr)
enddo

CC End of iteration
<table>
<thead>
<tr>
<th>Time%</th>
<th>Time</th>
<th>Imb. Time</th>
<th>Imb. Time%</th>
<th>Calls</th>
<th>Group</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td>6.289939</td>
<td>--</td>
<td>--</td>
<td>2.091</td>
<td>Total</td>
<td></td>
</tr>
<tr>
<td>72.5%</td>
<td>4.556013</td>
<td>--</td>
<td>--</td>
<td>384.0</td>
<td>OACC</td>
<td></td>
</tr>
<tr>
<td>55.7%</td>
<td>3.502276</td>
<td>0.043724</td>
<td>1.6%</td>
<td>93.0</td>
<td>jacobi_ACC_COPY@li.212</td>
<td></td>
</tr>
<tr>
<td>12.6%</td>
<td>0.787380</td>
<td>0.021668</td>
<td>3.6%</td>
<td>6.0</td>
<td>jacobi_ACC_COPY@li.89</td>
<td></td>
</tr>
<tr>
<td>3.6%</td>
<td>0.236862</td>
<td>0.008640</td>
<td>3.2%</td>
<td>93.0</td>
<td>jacobi_ACC_COPY@li.238</td>
<td></td>
</tr>
<tr>
<td>0.4%</td>
<td>0.023342</td>
<td>0.005695</td>
<td>2.6%</td>
<td>93.0</td>
<td>jacobi_ACC_ASYNC_KERNEL@li.212</td>
<td></td>
</tr>
<tr>
<td>0.1%</td>
<td>0.006408</td>
<td>0.001959</td>
<td>11.2%</td>
<td>93.0</td>
<td>jacobi_ACC_ASYNC_KERNEL@li.212</td>
<td></td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000592</td>
<td>0.000102</td>
<td>17.2%</td>
<td>3.0</td>
<td>jacobi_ACC_ASYNC_KERNEL@li.89</td>
<td></td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000450</td>
<td>0.000134</td>
<td>30.6%</td>
<td>3.0</td>
<td>jacobi_ACC_ASYNC_KERNEL@li.89</td>
<td></td>
</tr>
<tr>
<td>21.6%</td>
<td>1.353529</td>
<td>--</td>
<td>--</td>
<td>99.0</td>
<td>USER</td>
<td></td>
</tr>
<tr>
<td>18.8%</td>
<td>1.129580</td>
<td>0.024657</td>
<td>2.8%</td>
<td>1.0</td>
<td>himenobntxp_</td>
<td></td>
</tr>
<tr>
<td>2.6%</td>
<td>0.124014</td>
<td>0.008535</td>
<td>6.6%</td>
<td>1.0</td>
<td>initmnt_</td>
<td></td>
</tr>
<tr>
<td>1.5%</td>
<td>0.088615</td>
<td>0.001239</td>
<td>19.8%</td>
<td>1.0</td>
<td>jacobi_</td>
<td></td>
</tr>
<tr>
<td>0.6%</td>
<td>0.001281</td>
<td>0.000173</td>
<td>15.8%</td>
<td>93.0</td>
<td>jacobi_ACC_REGION@li.212</td>
<td></td>
</tr>
<tr>
<td>0.6%</td>
<td>0.000339</td>
<td>0.000009</td>
<td>25.2%</td>
<td>3.0</td>
<td>jacobi_ACC_REGION@li.89</td>
<td></td>
</tr>
<tr>
<td>4.4%</td>
<td>0.277417</td>
<td>--</td>
<td>--</td>
<td>430.0</td>
<td>ETC</td>
<td></td>
</tr>
<tr>
<td>4.4%</td>
<td>0.276967</td>
<td>0.000226</td>
<td>6.1%</td>
<td>17.0</td>
<td>__STOP3</td>
<td></td>
</tr>
<tr>
<td>6.0%</td>
<td>0.000121</td>
<td>0.000013</td>
<td>12.6%</td>
<td>18.0</td>
<td>cuDLOrigin</td>
<td></td>
</tr>
<tr>
<td>6.0%</td>
<td>0.000059</td>
<td>0.000007</td>
<td>12.6%</td>
<td>408.0</td>
<td><em>libc_origen</em></td>
<td></td>
</tr>
<tr>
<td>0.6%</td>
<td>0.000090</td>
<td>0.000001</td>
<td>15.5%</td>
<td>1.0</td>
<td>cudbMain</td>
<td></td>
</tr>
<tr>
<td>0.6%</td>
<td>0.000005</td>
<td>0.000000</td>
<td>19.9%</td>
<td>2.0</td>
<td>cuVDAUCtxCreate</td>
<td></td>
</tr>
<tr>
<td>1.5%</td>
<td>0.031660</td>
<td>--</td>
<td>--</td>
<td>1,076.0</td>
<td>MPI</td>
<td></td>
</tr>
<tr>
<td>1.1%</td>
<td>0.071594</td>
<td>0.054624</td>
<td>57.7%</td>
<td>192.0</td>
<td>MPI_WAITALL</td>
<td></td>
</tr>
<tr>
<td>0.2%</td>
<td>0.016993</td>
<td>0.002644</td>
<td>20.8%</td>
<td>384.0</td>
<td>MPI_SEND</td>
<td></td>
</tr>
<tr>
<td>0.1%</td>
<td>0.004359</td>
<td>0.000047</td>
<td>22.6%</td>
<td>384.0</td>
<td>MPI_RECV</td>
<td></td>
</tr>
<tr>
<td>0.6%</td>
<td>0.000397</td>
<td>0.000015</td>
<td>4.9%</td>
<td>97.0</td>
<td>MPI_ALLREDUCE</td>
<td></td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000053</td>
<td>0.000003</td>
<td>6.7%</td>
<td>1.0</td>
<td>MPI_CART_CREATE</td>
<td></td>
</tr>
</tbody>
</table>

Hewlett Packard Enterprise
Complications with Himeno

• All Arrays are in COMMON blocks – need to be global

• Reveal is putting data transfer inside the iteration loop

  • However, it does recognize that on P and WRK2 need to be updated

• Biggest issue is transferring the entire P array when only boundaries need to be exchanged

• Following charts give some optimizations for himeno that do not deal with halo exchange of P
  • The plan is to put all the arrays and operations on those arrays on the accelerator.
  • Whenever the host needs to do something with the accelerator data, we must update the host from the accelerator
Initialize arrays on the accelerator

```fortran
148.                                                                                         !******************************************************************************
149.                                                                                         subroutine initmt(mz,it)
150.                                                                                         !******************************************************************************
151.                                                                                         IMPLICIT REAL*4(a-h,o-z)
152.                                                                                         C
153.                                                                                         include 'param.h'
154.                                                                                         C
155.                                                                                         !$OMP TARGET DATA map(alloc:a,b,c,bnd,wrk1,wrk2,p)
156.                                                                                         !$OMP target teams distribute
157.                                                                                         do k=1,mkmax
158.                                                                                         do j=1,mjmax
159.                                                                                         do i=1,mimax
160.                                                                                         a(i,j,k,1)=0.0
161.                                                                                         a(i,j,k,2)=0.0
162.                                                                                         a(i,j,k,3)=0.0
163.                                                                                         a(i,j,k,4)=0.0
164.                                                                                         b(i,j,k,1)=0.0
165.                                                                                         b(i,j,k,2)=0.0
166.                                                                                         b(i,j,k,3)=0.0
167.                                                                                         b(i,j,k,4)=0.0
168.                                                                                         c(i,j,k,1)=0.0
169.                                                                                         c(i,j,k,2)=0.0
170.                                                                                         c(i,j,k,3)=0.0
171.                                                                                         c(i,j,k,4)=0.0
172.                                                                                         p(i,j,k)=0.0
173.                                                                                         wrk1(i,j,k)=0.0
174.                                                                                         wrk2(i,j,k)=0.0
175.                                                                                         bnd(i,j,k)=0.0
176.                                                                                         enddo
177.                                                                                         enddo
178.                                                                                         enddo
```
206. subroutine jacobi(nn,gosa)
207. c********************************************************************************
208. implicit real*4(a-h,o-z)
209. c
210. include 'mpif.h'
211. include 'param.h'
212. c
213. + 1-----------------< do loop=1,nn
214. 1 gosa=0.0
215. 1 wgosa=0.0
216. + 1 !$omp target data map(tofrom:p)
217. 1 ! Directive inserted by Cray Reveal. May be incomplete.
218. + 1 MG---------------< !$omp target teams distribute
219. 1 MG !$omp& private(i, j, k, s0, ss)
220. 1 MG !$omp& firstprivate(imax, jmax, kmax, omega)
221. 1 MG !$omp& map(tofrom: wgosa)
222. 1 MG !$omp& reduction(+: wgosa)
223. 1 MG g-------< do k=2,kmax-1
224. 1 MG g 4--------< do j=2,jmax-1
225. 1 MG g 4 gr3----< do i=2,imax-1
226. 1 MG g 4 gr3 1 so=a(i,j,k,1)*p(i+1,j,k)+a(i,j,k,2)*p(i,j+1,k)
227. 1 MG g 4 gr3 2 +a(i,j,k,3)*p(i,j+1,k)
228. 1 MG g 4 gr3 3 +b(i,j,k,1)*p(i+1,j+1,k)-p(i+1,j-1,k)
229. 1 MG g 4 gr3 4 -p(i-1,j+1,k)+p(i-1,j-1,k))
230. 1 MG g 4 gr3 5 +b(i,j,k,2)*p(i,j+1,k+1)-p(i,j,k+1)
231. 1 MG g 4 gr3 6 -p(i,j+1,k+1)+p(i,j,k+1))
232. 1 MG g 4 gr3 7 +b(i,j,k,3)*p(i+1,j,k+1)-p(i-1,j,k+1)
233. 1 MG g 4 gr3 8 -p(i+1,j,k+1)+p(i-1,j,k+1))
234. 1 MG g 4 gr3 9 +c(i,j,k,1)*p(i-1,j,k)+c(i,j,k,2)*p(i,j-1,k)
235. 1 MG g 4 gr3 10 +c(i,j,k,3)*p(i,j,k-1)+wrk1(i,j,k)
236. 1 MG g 4 gr3 ss=(so*a(i,j,k,4)-p(i,j,k))*bnd(i,j,k)
237. 1 MG g 4 gr3 wgosa=wgosa+ss*ss
238. 1 MG g 4 gr3 wrk2(i,j,k)=p(i,j,k)+omega *ss
239. 1 MG g 4 gr3 wrk3(i,j,k)=wrk2(i,j,k)**2
240. 1 MG g 4 gr3 wrk3----> enddo
241. 1 MG g 4 gr3 enddo
242. c
242. 1  
243. + 1 MG----------< !$OMP target teams distribute  
244. 1 MG  
245. + 1 MG  
246. 1 MG g----------<  
247. + 1 MG  
248. 1 MG g  
249. 1 MG g  
250. 1 MG g  
251. 1 MG g  
252. 1 MG g---------->  
253. 1  
254. + 1 MG----------<  
255. + 1 E----------<  
256. + 1 
257. + 1  
258. 1  
259. 1  
260. 1  
261. 1  
262. 1  
263. 1  
264. 1  
265. 1----------->  
266. CC End of iteration  
267. return  
268. end

!$OMP target teams distribute

!$OMP& private(i, j, k)

!$OMP& firstprivate(imax, jmax, kmax)

DO K=2,kmax-1

DO J=2,jmax-1

DO I=2,imax-1

p(I,J,K)=wrk2(I,J,K)

enddo

enddo

enddo

!$OMP END TARGET DATA

call sendp(ndx,ndy,ndz)

call mpi_allreduce(wgos, gosa, i)

call mpi_allreduce(gosa, mpi_real4, mpi_sum, mpi_comm_world, ierr)

enddo

CC End of iteration

return

end
At this point I am moving to illustrate halo exchanges on a different application – leslie3d

• Himeno uses MPI message types which are more confusing

• Leslie3d packs and unpacks buffers and it is easier to illustrate optimizing on the GPU
One of the eight buffers required for 3-D application

IF (NCY .LT. NPY .OR. (NCY .EQ. NPY .AND. JPERIODIC)) THEN
    JCNT = 0
    DO K = 1-NLEVELS, KCMAK+NLEVELS
        DO J = JCMAX-NLEVELS+1, JCMAX
            DO I = 1-NLEVELS, ICMAX+NLEVELS
                JCNT = JCNT + 1
                S_N(JCNT) = VAR(I,J,K)
            END DO
        END DO
    END DO
END DO
CALL MPI_SEND(S_N, NLEVELS * NSIZE1 * NSIZE3, MPI_DOUBLE_PRECISION, NORTH, MSGFLAG + MSGTSN, MPI_COMM_WORLD, IERR )
IF ( IERR .NE. 0 ) CALL EJECT ('MPI_SEND FAILED: S_N')

Need to pack buffers on the accelerator
Packed on the GPU and sending the halo

```c
#ifdef OMP_TARGET
G---------------
#endif
G g---------------
DO K = 1, NLEVELS
+ G g 3---------------
DO J = 1-NLEVELS, JCMAX+NLEVELS
+ G g 3
!$omp target teams distribute
G g 3
#ifdef OMP_TARGET
!$
omp parallel do private(kcnt) private(i)
G g 3
#endif
G g 3 g--------------
DO I = 1-NLEVELS, ICMAX+NLEVELS
G g 3 g
KCNT = (k-1)*(JCMAX+2*NLEVELS)*(ICMAX+2*NLEVELS)
G g 3 g
> +(j-1+NLEVELS)*(ICMAX+2*NLEVELS)+i
G g 3 g
END DO
G g 3 g
S_I(KCNT) = VAR(I,J,K)
G g 3 g
ENDD0
G g 3 g
END DO
G
!$omp end target teams distribute
G
#ifdef OMP_TARGET
!$
omp target update from(S_I)
G
#endif
CALL MPI_SEND(S_I,
> NLEVELS * NSIZE1 * NSIZE2,
> MPI_DOUBLE_PRECISION,
> IN,
> MSGFLAG + MSGTSI,
> MPI_COMM_WORLD,
> ierr )
IF ( ierr .NE. 0 ) CALL EJECT ('MPI_SEND FAILED: S_I')
END IF
```

We do have to update the host prior to the MPI_SEND.
Posting the receive on the Host

We don’t have to do anything prior to or after the IRECV

```
706.707.  IF(NCZ .GT. 1 .OR. (NCZ .EQ. 1 .AND. KPERIODIC)) THEN
708.      CALL MPI_IRECV(R_I,
709.         > NLEVELS * NSIZE1 * NSIZE2,
710.         > MPI_DOUBLE_PRECISION,
711.         > IN,
712.         > MSGFLAG + MSGTRI,
713.         > MPI_COMM_WORLD,
714.         > MSGIDRI,
715.         > IERR )
```
After the MPI_WAIT we need to update the GPU with the communicated array.

Receiving and unpacking on the GPU

CALL MPI_WAIT( MSGIDRI, MPI_STATUS, IERR)

ifdef OMP_TARGET
!$omp target update to(R_I)
endif

DO K = 1-NLEVELS, 0
    DO J = 1-NLEVELS, JCMAX+NLEVELS
        KCNT = (k-1+NLEVELS)*(JCMAX+2*NLEVELS)*(ICMAX+2*NLEVELS)
        + (j-1+NLEVELS)*(ICMAX+2*NLEVELS)+i
    END DO
    VAR(I,J,K) = R_I(KCNT)
END DO

ifdef OMP_TARGET
!$omp end target teams distribute
endif
Still can do better – use use_device_ptr and setenv MPICH_RDMA_ENABLED_CUDA 1

```c
#include OMP_TARGET
"omp target enter data map(to:VAR)
"omp& map(alloc:R_N,R_W,R_S,R_E,R_I,R_O,S_N,S_W,S_S,
"omp& S_E,S_I,S_O)
"omp& use_device_ptr(R_N,R_W,R_S,R_E,R_I,R_O,S_N,S_W,S_S,
"omp& S_E,S_I,S_O)
#endif
```

CALL MPI_IRECV(R_W,
    NLEVELS * NSIZE2 * NSIZE3,
    MPI_DOUBLE_PRECISION,
    WEST,
    MSGFLAG + MSGTRW,
    MPI_COMM_WORLD,
    MSGIDRW,
    IERR)

Much easier to code –
Do not need target update clauses
ifdef OMP_TARGET
!$omp target enter data map(to:VAR)
!$omp&      map(alloc:R_N,R_W,R_S,R_E,R_I,R_O,S_N,S_W,S_S,
!$OMP&        S_E,S_I,S_O)
!$omp target data
!$omp& use_device_ptr(R_N,R_W,R_S,R_E,R_I,R_O,S_N,S_W,S_S,
!$OMP&        S_E,S_I,S_O)
#endif

CALL MPI_SEND(S_E,
>     NLEVELS * NSIZE2 * NSIZE3,
>     MPI_DOUBLE_PRECISION,
>     EAST,
>     MSGFLAG + MSGTSE,
>     MPI_COMM_WORLD,
>     IERR )

Much easier to code
All Packing done on the GPU

Comparisons Using Device Buffers or Host Buffers

Wallclock Seconds

Number of Nodes (1 GPU/Node)

- Use host Buffers
- Use Device Pointers
CRAY_ACC_DEBUG Environment Variable

- Three levels of verbosity
  - 1 High Level overview of kernels executed and data transferred
  - 2 Breaks down data transfer by each variable
  - 3 The whole Kitchen sink
setenv CRAY_ACC_DEBUG 1

ACC: Transfer 29 items (to acc 0 bytes, to host 0 bytes) from src/fluxi.f:21
ACC: Transfer 15 items (to acc 0 bytes, to host 0 bytes) from src/fluxi.f:114
ACC: Transfer 1 items (to acc 0 bytes, to host 0 bytes) from src/fluxi.f:141
ACC: Transfer 29 items (to acc 0 bytes, to host 0 bytes) from src/fluxi.f:21
ACC: Transfer 15 items (to acc 0 bytes, to host 0 bytes) from src/fluxi.f:114
ACC: Transfer 1 items (to acc 0 bytes, to host 0 bytes) from src/fluxi.f:141
ACC: Execute kernel extrapi_$ck_L141_7 async(auto) from src/fluxi.f:141
ACC: Execute kernel extrapi_$ck_L141_7 async(auto) from src/fluxi.f:141
ACC: Wait async(auto) from src/fluxi.f:172
ACC: Transfer 1 items (to acc 0 bytes, to host 0 bytes) from src/fluxi.f:172
ACC: Execute kernel extrapi_$ck_L173_9 async(auto) from src/fluxi.f:173
ACC: Wait async(auto) from src/fluxi.f:172
setenv CRAY_ACC_DEBUG 2

ACC: Start transfer 29 items from src/fluxi.f:21
ACC: present 'dq' (255923712 bytes)
ACC: present 'ds' (95971392 bytes)
ACC: present 'ds1' (95971392 bytes)
ACC: present 'dtv' (31990464 bytes)
ACC: present 'hf' (127961856 bytes)
ACC: present 'pav' (31990464 bytes)
ACC: present 'q' (511847424 bytes)
ACC: present 'qav' (255923712 bytes)
ACC: present 'six' (31990464 bytes)
ACC: present 'siy' (31990464 bytes)
ACC: present 'siz' (31990464 bytes)
ACC: present 't' (31990464 bytes)
setenv CRAY_ACC_DEBUG 3

ACC: Start transfer 29 items from src/fluxi.f:21
ACC: flags: RETURN_ACC_TIME
ACC: Trans 1
ACC: Simple transfer of 'dq' (144 bytes)
ACC: host ptr 78b538
ACC: acc ptr 0
ACC: flags: DOPE_VECTOR DV_ONLY_DATA ALLOCATE COPY_HOST_TO_ACC ACQ_PRESENT REG_PRESENT
ACC: Transferring dope vector
ACC: dim:1 lowbound:-2 extent:198 stride_mult:1
ACC: dim:2 lowbound:-2 extent:198 stride_mult:198
ACC: dim:3 lowbound:-2 extent:102 stride_mult:39204
ACC: dim:4 lowbound:1 extent:8 stride_mult:3998808
ACC: DV size=255923712 (scale:8, elem_size:8)
ACC: total mem size=255923712 (dv:0 obj:255923712)
ACC: host region 4dac8780 to 5ced9d80 found in present table index 0 (ref count 2)
ACC: memory found in present table (2aaaf2000000, base 2aaaf2000000)
ACC: new acc ptr 2aaaf2000000
So what have we learned

• Perftools is excellent for identifying issues in existing applications for improving threading, vectorization and scalar optimization
• Reveal can help with the difficult job of scoping variables in potential parallelizable loops
  • More difficult if not impossible with C++
• Moving to the GPU is difficult; however, it can be done in steps that are more manageable
  • Perftools identifies the bottlenecks in the GPU application very quickly
• GPU direct is best way to do the message passing – in this case it only matters at scale
Thank you