Preparing an application for Hybrid Supercomputing

John Levesque, Harvey Richardson, Alfio Lazzaro, Nina Mujkanovic, HPE

Using HPE Programming Environment for GPUs - DEC 2023
Agenda

- Introduction
- Steps in moving an application to a GPU
- A simple example – himeno
- Optimizing himeno – with OpenMP on the node
- Optimizing himeno – with OpenMP Offload (or OpenACC) for GPU
- Dealing with data movement
  - CPU <-> GPU
  - Halo Exchanges
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
- Abnormal Termination Processing
  Manage core files at scale

PERFORMANCE ANALYSIS, PORTING, AND OPTIMIZATION

- Performance Analysis Tools
  Simple and advanced interfaces provide whole program profiling + visualization
- Code Parallelization Assistant
  Reveal hidden potential of an application via code restructuring
- Comparative Debugger
  Compare two versions of an application
- STAT
  Stack trace analysis at scale
- GDB for HPC
  Parallel gdb for scalable debugging
- Valgrind for HPC
  Memory debugging at scale

DEBUGGING

© Copyright 2023 Hewlett Packard Enterprise Development LP
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>HPE Cray Programming Environment</th>
<th>AMD Programming Environment</th>
<th>Intel Programming Environment</th>
<th>NVIDIA Programming Environment</th>
<th>GNU Programming Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiling Environment (CCE)</td>
<td>AMD AOCC and ROCm compilers</td>
<td>Intel® C, C++, and Fortran compilers</td>
<td>NVIDIA compilers</td>
<td>GNU Compiler Collection</td>
</tr>
<tr>
<td>Cray MPI and SHMEM</td>
<td>Performance Analysis Tools</td>
<td>Debugger Support Tools</td>
<td>Scientific and Math Libraries</td>
<td>Environment Setup and Compiling Support</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Third Party Products</td>
</tr>
</tbody>
</table>

© Copyright 2023 Hewlett Packard Enterprise Development LP
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
- OpenMP offload and OpenACC support features to integrate with models such as CUDA and HIP
  - 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)
Himeno benchmark

- 3D Poisson equation
  - 19-point stencil
  - Highly memory intensive, memory bound
- Fortran, C, MPI and OpenMP versions
- Variously sized configurations
Himeno Benchmark kernel

- The stencil is applied to pressure array $p$
- Updated pressure values are saved to temporary array $wrk2$
- Control value $wgosa$ is computed
- In the benchmark this kernel is iterated a fixed number of times ($nn$)
Distributed version of Himeno

- The outer loop is performed a fixed number of times
- The Jacobi kernel is executed and new pressure array \( wrk2 \) and control value \( wgosa \) are computed
- The array is updated with the new pressure values
- The halo region values are exchanged between neighbor PEs
- Send and receive buffers are used
- The maximum control value is computed with an Allreduce operation across all the PEs

```plaintext
DO loop = 1, nn
    compute Jacobi kernel \( \rightarrow wrk2, wgosa \)
       copy back \( wrk2 \) into \( p \)
       pack halo from \( p \) into send buffers
       exchange halos with neighbour PEs
       unpack halo into \( p \) from recv buffers
    Allreduce to sum \( wgosa \) across PEs
ENDDO
```
Using perftools-lite ( or perftools-lite-loops or perftools-lite-hbm )

- Perftools-lite modules are the simplest way to run a performance experiment

- Before building an application:
  - 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 is output to standard output at end of application execution
  - Also generates a directory of profile data which can be used to generate reports with different options
Let’s start with a simple example

% cd F_mpi
% module load perftools-lite

% ftn -O3 -rm -hpl=himeno.pl -I. -c himeno.f90 -o himeno.o
% ftn -O3 -rm -hpl=himeno.pl himeno.o -o himeno.exe
INFO: creating the PerfTools-instrumented executable 'himeno.exe' (lite-samples) ...OK

% cat job.slurm
#!/bin/bash
#SBATCH -t 10
#SBATCH -N 1
#SBATCH --ntasks-per-node=8
#SBATCH --exclusive

DIST="--cpu-bind=mask_cpu:0xfe,0xfe00,0xfe00000,0xfe0000000,0xfe000000000,0xfe00000000000,0xfe0000000000000,0xfe00000000000000"

export LD_LIBRARY_PATH=${CRAY_LD_LIBRARY_PATH}:${LD_LIBRARY_PATH}

time srun ${DIST} himeno.exe+orig > my_output_orig.$(SLURM_JOBID) 2>&1

time srun ${DIST} himeno.exe > my_output.$(SLURM_JOBID) 2>&1

% sbatch job.slurm

Files after run
himeno.exe+125664-8747781s himeno.lst my_output.3458017 my_output_orig.3458017 slurm-3458017.out
Himeno output

% cat my_output.*
CrayPat/X: Version 23.03.0 Revision 46f710008 02/13/23 20:24:04
Sequential version array size
  mimax= 1025  mjmax= 513  mkmax= 513
Parallel version  array size
  mimax= 515  mjmax= 259  mkmax= 259
  imax= 513  jmax= 257  kmax= 257
  I-decomp= 2  J-decomp= 2  K-decomp= 2

Start rehearsal measurement process.
Measure the performance in 10 times.
  MFLOPS: 69617.461462995881  time(s): 1.2982310590000452, 4.340651794E-4
Now, start the actual measurement process.
The loop will be executed in 50 times.
This will take about one minute.
Wait for a while.
  cpu  6.467868152000392 sec
  Loop executed for 50 times
  Gosa : 4.232053179E-4
  MFLOPS measured : 69868.11465231188
  Score based on Pentium III 600MHz : 843.4104
## Perftools-lite profile

### 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>
<tr>
<td>100.0%</td>
<td>841.0</td>
<td>--</td>
<td>--</td>
<td>Total</td>
</tr>
<tr>
<td>80.8%</td>
<td>679.6</td>
<td>--</td>
<td>--</td>
<td>USER</td>
</tr>
<tr>
<td>75.8%</td>
<td>637.5</td>
<td>3.5</td>
<td>0.6%</td>
<td>jacobi_</td>
</tr>
<tr>
<td>5.0%</td>
<td>42.1</td>
<td>3.9</td>
<td>9.6%</td>
<td>himenobmtxp_</td>
</tr>
<tr>
<td>17.0%</td>
<td>142.9</td>
<td>--</td>
<td>--</td>
<td>ETC</td>
</tr>
<tr>
<td>14.1%</td>
<td>119.0</td>
<td>3.0</td>
<td>2.8%</td>
<td>__cray_memcpy_ROME</td>
</tr>
<tr>
<td>2.8%</td>
<td>23.2</td>
<td>4.8</td>
<td>19.4%</td>
<td>__cray_memset_ROME</td>
</tr>
<tr>
<td>2.2%</td>
<td>18.5</td>
<td>--</td>
<td>--</td>
<td>MPI</td>
</tr>
<tr>
<td>1.3%</td>
<td>10.9</td>
<td>6.1</td>
<td>41.2%</td>
<td>MPI_WAITALL</td>
</tr>
</tbody>
</table>

- Exclusive time
- Sampling is in 100th of a second
- Imbalance
- Only showing items that take up more than 1% of time – you can override with -T in pat_report
# Perftools-lite profile

<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>841.0</td>
<td>--</td>
<td>--</td>
<td>Total</td>
</tr>
</tbody>
</table>

| 80.8% | 679.6 | -- | -- | USER |

| 75.8% | 637.5 | 3.5 | 0.6% | jacobi_ |
| 5.0%  | 42.1  | 3.9 | 9.6% | himenobmtxp_ |

| 17.0% | 142.9 | -- | -- | ETC |

| 14.1% | 119.0 | 3.0 | 2.8% | __cray_memcpy_ROME |
| 2.8%  | 23.2  | 4.8 | 19.4% | __cray_memset_ROME |
| 0.0%  | 0.2   | 0.8 | 85.7% | __cray_scopy_detect |
| 0.0%  | 0.1   | 0.9 | 100.0% | _init |
| 0.0%  | 0.1   | 0.9 | 100.0% | __cray_sset_ROME |
| 0.0%  | 0.1   | 0.9 | 100.0% | __cray_scopy_ROME |

| 2.2%  | 18.5  | -- | -- | MPI |

| 1.3%  | 10.9  | 6.1 | 41.2% | MPI_WAITALL |
| 0.7%  | 5.5   | 1.5 | 24.5% | MPI_ISEND |
| 0.2%  | 1.5   | 1.5 | 57.1% | MPI_IRecv |
| 0.1%  | 0.5   | 0.5 | 57.1% | MPI_BARRIER |
| 0.0%  | 0.1   | 0.9 | 100.0% | mpi_recv_

---

```
pat_report -T himeno.exe+125664-8747781s > profile_T
```

**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></td>
<td></td>
<td></td>
<td></td>
<td>PE=[max,min]</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>100.0%</td>
<td>641.0</td>
<td>3.5</td>
<td>0.6%</td>
<td>jacob<strong>i_</strong></td>
</tr>
<tr>
<td>100.0%</td>
<td>641.0</td>
<td>--</td>
<td>--</td>
<td>pe.3</td>
</tr>
<tr>
<td>98.4%</td>
<td>631.0</td>
<td>--</td>
<td>--</td>
<td>pe.6</td>
</tr>
<tr>
<td>19.0%</td>
<td>122.0</td>
<td>3.0</td>
<td>2.8%</td>
<td>__cray_memcpy_ROME</td>
</tr>
<tr>
<td>19.0%</td>
<td>122.0</td>
<td>--</td>
<td>--</td>
<td>pe.0</td>
</tr>
<tr>
<td>18.3%</td>
<td>117.0</td>
<td>--</td>
<td>--</td>
<td>pe.5</td>
</tr>
<tr>
<td>7.2%</td>
<td>46.0</td>
<td>3.9</td>
<td>9.6%</td>
<td>himenobmtxp_</td>
</tr>
<tr>
<td>7.2%</td>
<td>46.0</td>
<td>--</td>
<td>--</td>
<td>pe.3</td>
</tr>
<tr>
<td>5.9%</td>
<td>38.0</td>
<td>--</td>
<td>--</td>
<td>pe.6</td>
</tr>
<tr>
<td>4.4%</td>
<td>28.0</td>
<td>4.8</td>
<td>19.4%</td>
<td>__cray_memset_ROME</td>
</tr>
<tr>
<td>4.4%</td>
<td>28.0</td>
<td>--</td>
<td>--</td>
<td>pe.1</td>
</tr>
<tr>
<td>3.0%</td>
<td>19.0</td>
<td>--</td>
<td>--</td>
<td>pe.4</td>
</tr>
</tbody>
</table>

...
## Perftools-lite profile

---

### Total

```plaintext
<table>
<thead>
<tr>
<th>Category</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread Time</td>
<td>51.414680 secs</td>
</tr>
<tr>
<td>UNHALTED_REFERENCE_CYCLES</td>
<td>117,110,852,571</td>
</tr>
<tr>
<td>CPU_CLK_THREAD_UNHALTED:THREAD_P</td>
<td>181,647,161,792</td>
</tr>
<tr>
<td>INST_RETIRED:ANY_P</td>
<td>109,619,182,320</td>
</tr>
<tr>
<td>RESOURCESTALLS:ANY</td>
<td>145,947,632,023</td>
</tr>
<tr>
<td>FP_ARITH:PACKED</td>
<td>38,585,536,000</td>
</tr>
<tr>
<td>OFFCORE_RESPONSE_0:ANY_REQUEST:L3_MISS_LOCAL</td>
<td>4,480,911,135</td>
</tr>
<tr>
<td>CPU CLK Boost</td>
<td>1.55 X</td>
</tr>
<tr>
<td>Resource stall cycles / Cycles</td>
<td>80.3%</td>
</tr>
<tr>
<td>FP Packed Instr / All Instr</td>
<td>35.2%</td>
</tr>
<tr>
<td>Memory traffic GBytes</td>
<td>5.578G/sec</td>
</tr>
<tr>
<td>Local Memory traffic GBytes</td>
<td>286.78 GB</td>
</tr>
<tr>
<td>Memory Traffic / Nominal Peak</td>
<td>7.3%</td>
</tr>
<tr>
<td>Retired Inst per Clock</td>
<td>0.60</td>
</tr>
</tbody>
</table>
```
Perftools-lite profile

Notes for table 5:

This table show 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 MiBytes per Writer Rank</th>
<th>Write Rate MiBytes/sec</th>
<th>Number of Writer Ranks</th>
<th>Avg Writes per Writer Rank</th>
<th>Avg Bytes/Call</th>
<th>File Name PE=HIDE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000014</td>
<td>0.000008</td>
<td>0.540886</td>
<td>8</td>
<td>1.0</td>
<td>8.00</td>
<td>stderr</td>
</tr>
<tr>
<td>0.000004</td>
<td>0.0000628</td>
<td>152.051781</td>
<td>1</td>
<td>19.0</td>
<td>34.63</td>
<td>stdout</td>
</tr>
</tbody>
</table>
## Perftools-lite profile

Table 3: Profile by Group, Function, and Line

<table>
<thead>
<tr>
<th>Group</th>
<th>Source</th>
<th>Line</th>
<th>PE=HIDE</th>
<th>jacobi_3</th>
<th>cug23/himeno_new/F_mpi/himeno.f90</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td>100.0%</td>
<td>841.0</td>
<td></td>
</tr>
<tr>
<td>USER</td>
<td></td>
<td></td>
<td>80.8%</td>
<td>679.6</td>
<td></td>
</tr>
<tr>
<td>USER</td>
<td></td>
<td></td>
<td>75.8%</td>
<td>637.5</td>
<td></td>
</tr>
<tr>
<td>USER</td>
<td></td>
<td></td>
<td>4</td>
<td>46.8%</td>
<td>393.4</td>
</tr>
<tr>
<td>USER</td>
<td></td>
<td></td>
<td>4</td>
<td>15.9%</td>
<td>133.5</td>
</tr>
<tr>
<td>USER</td>
<td></td>
<td></td>
<td>4</td>
<td>0.6%</td>
<td>5.4</td>
</tr>
<tr>
<td>USER</td>
<td></td>
<td></td>
<td>4</td>
<td>11.2%</td>
<td>94.0</td>
</tr>
<tr>
<td>USER</td>
<td></td>
<td></td>
<td>4</td>
<td>0.2%</td>
<td>1.8</td>
</tr>
<tr>
<td>USER</td>
<td></td>
<td></td>
<td>4</td>
<td>0.0%</td>
<td>0.1</td>
</tr>
<tr>
<td>USER</td>
<td></td>
<td></td>
<td>3</td>
<td>5.0%</td>
<td>42.1</td>
</tr>
<tr>
<td>USER</td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>USER</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© Copyright 2023 Hewlett Packard Enterprise Development LP
SUBROUTINE jacobi(nn,gosa)
  IMPLICIT REAL*4(a-h,o-z)
  INTEGER, INTENT(IN) :: nn
  REAL, INTENT(OUT)  :: gosa
  INTEGER*4 istat
  INCLUDE 'mpif.h'
  INCLUDE 'param.h'

+ 1-----------------< DO loop=1 nn
  1 gosa=0.0
  1 wgosa=0.0
  1
+ 1 2-----------------< DO k=2,kmax-1
+ 1 2 3-----------------< DO j=2,jmax-1
  1 2 3 Vr3------------< DO i=2,imax-1

  1 2 3 Vr3      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) &
  1 2 3 Vr3 +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)) &
  1 2 3 Vr3 +b(i,j,k,2)*(p(i+1,j+1,k)-p(i+1,j-1,k) -p(i-1,j+1,k)+p(i-1,j-1,k)) &
  1 2 3 Vr3 +b(i,j,k,3)*(p(i+1,j,k+1)-p(i+1,j,k-1)-p(i+1,j,k+1)+p(i+1,j,k-1)) &
  1 2 3 Vr3 +c(i,j,k,1)*p(i-1,j,k)+c(i,j,k,2)*p(i,j-1,k) &
  1 2 3 Vr3 +c(i,j,k,3)*p(i,j-1,k)+wrk1(i,j,k)
  1 2 3 Vr3      ss=(s0*a(i,j,k,4)-p(i,j,k))*bnd(i,j,k)
  1 2 3 Vr3      wgosa=wgosa+ss*ss
  1 2 3 Vr3      wrk2(i,j,k)=p(i,j,k)+omega *ss
  1 2 3 Vr3--------> END DO
  1 2 3 Vr3--------> END DO
  1 2 ----------------< END DO

**Vectorised and unrolled by 3**
Compiler loopmark information...

235. 1
236. + 1 2------------------< DO k=2,kmax-1
237. + 1 2 3----------------< DO j=2,jmax-1
238. 1 2 3 A---------------< DO i=2,imax-1
239. 1 2 3 A
   p(i,j,k)=wrk2(i,j,k)
240. 1 2 3 A--------------> END DO
241. 1 2 3---------------> END DO
242. 1 2----------------> END DO
243. 1
244. 1 I
   CALL sendp(ndx,ndy,ndz)
245. 1
246. + 1
   CALL mpi_allreduce(wgosa, gosa, &
247. 1
   1, mpi_real4, &
248. 1
   mpi_sum, mpi_comm_world, &
249. 1
   ierr)
250. 1
251. 1-----------------> END DO
252.
   ! End of iteration
253.
   RETURN
254.
END SUBROUTINE jacobi
## Loopmark Legend

<table>
<thead>
<tr>
<th>Primary Loop Type</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Pattern matched</td>
<td>a Atomic memory operation</td>
</tr>
<tr>
<td></td>
<td>b Blocked</td>
</tr>
<tr>
<td>C Collapsed</td>
<td>c Conditional and/or computed</td>
</tr>
<tr>
<td>D Deleted</td>
<td></td>
</tr>
<tr>
<td>E Closed</td>
<td></td>
</tr>
<tr>
<td>F Flat – No calls</td>
<td>f Fused</td>
</tr>
<tr>
<td>G Accelerated</td>
<td>g Partitioned</td>
</tr>
<tr>
<td>I Inlined</td>
<td>i Interchanged</td>
</tr>
<tr>
<td>M Multithreaded</td>
<td>m Partitioned</td>
</tr>
<tr>
<td></td>
<td>n Non-blocking remote transfer</td>
</tr>
<tr>
<td></td>
<td>p Partial</td>
</tr>
<tr>
<td>R Rerolling</td>
<td>r Unrolled</td>
</tr>
<tr>
<td></td>
<td>s Shortloop</td>
</tr>
<tr>
<td>V Vectorized</td>
<td>w Unwound</td>
</tr>
</tbody>
</table>
Let’s Optimize this code

- Shared memory parallelism
- Then later move to the GPU
- To do these things we need to understand the loop limits (we want to go for loops that matter)

% module load perftools-lite-loops
% ftn -O3 -rm -hpl=himeno.pl -I. -c himeno.f90 -o himeno.o
% ftn -O3 -rm -hpl=himeno.pl himeno.o -o himeno.exe
INFO: creating the PerfTools-instrumented executable 'himeno.exe' (lite-loops) ...OK
$ sbatch job.slurm

Files after run
himeno.exe+27714-8747979t himeno.lst my_output.3458104
my_output_orig.3458104 slurm-3458104.out
### Perftools-lite-loops profile

Table 1: Calltree with Loop Inclusive Time

<table>
<thead>
<tr>
<th>Incl</th>
<th>Incl</th>
<th>Loop Exec</th>
<th>Loop</th>
<th>Calltree</th>
<th>Time%</th>
<th>Time</th>
<th>Trips</th>
<th>PE=HIDE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0%</td>
<td>8.38</td>
<td>--</td>
<td>--</td>
<td>total</td>
<td>100.0%</td>
<td>8.38</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>92.4%</td>
<td>7.74</td>
<td>--</td>
<td>--</td>
<td>himenobmtxtp_</td>
<td>92.4%</td>
<td>7.74</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>92.4%</td>
<td>7.74</td>
<td>--</td>
<td>--</td>
<td>jacobi_</td>
<td>92.4%</td>
<td>7.74</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>74.5%</td>
<td>6.25</td>
<td>2</td>
<td>30.0</td>
<td>jacobi_.LOOP.1.li.216</td>
<td>74.5%</td>
<td>6.25</td>
<td>2</td>
<td>30.0</td>
</tr>
<tr>
<td>74.5%</td>
<td>6.25</td>
<td>15,300</td>
<td>255.0</td>
<td>jacobi_.LOOP.2.li.220</td>
<td>74.5%</td>
<td>6.25</td>
<td>15,300</td>
<td>255.0</td>
</tr>
<tr>
<td>74.0%</td>
<td>6.20</td>
<td>3,901,500</td>
<td>511.0</td>
<td>jacobi_.LOOP.4.li.222</td>
<td>74.0%</td>
<td>6.20</td>
<td>3,901,500</td>
<td>511.0</td>
</tr>
<tr>
<td>14.6%</td>
<td>1.22</td>
<td>60</td>
<td>255.0</td>
<td>jacobi_.LOOP.5.li.236</td>
<td>14.6%</td>
<td>1.22</td>
<td>60</td>
<td>255.0</td>
</tr>
<tr>
<td>14.6%</td>
<td>1.22</td>
<td>15,300</td>
<td>255.0</td>
<td>jacobi_.LOOP.6.li.237</td>
<td>14.6%</td>
<td>1.22</td>
<td>15,300</td>
<td>255.0</td>
</tr>
<tr>
<td>9.7%</td>
<td>0.82</td>
<td>3,901,500</td>
<td>511.0</td>
<td>jacobi_.LOOP.7.li.238</td>
<td>9.7%</td>
<td>0.82</td>
<td>3,901,500</td>
<td>511.0</td>
</tr>
</tbody>
</table>

Inclusive time now since we are tracing and not sampling.

pat_report -T –Oct himeno.exe+27714-8747979t
Perf tools lite- **loops profile**... **pat_report -T -Oct himeno.exe+27714-8747979t**

<table>
<thead>
<tr>
<th>Incl</th>
<th>Incl</th>
<th>Loop Exec</th>
<th>Loop</th>
<th>Calltree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time%</td>
<td>Time</td>
<td>Trips</td>
<td>PE=HIDE</td>
<td>Avg</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>--------</td>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>7.5%</td>
<td>0.63</td>
<td>--</td>
<td>--</td>
<td>initmt_</td>
</tr>
<tr>
<td>5.3%</td>
<td>0.44</td>
<td>1</td>
<td>259.0</td>
<td>initmt_LOOP.1.11.156</td>
</tr>
<tr>
<td>5.3%</td>
<td>0.44</td>
<td>259</td>
<td>259.0</td>
<td>initmt_LOOP.2.11.157</td>
</tr>
<tr>
<td>5.2%</td>
<td>0.44</td>
<td>67,081</td>
<td>515.0</td>
<td>initmt_LOOP.3.11.158</td>
</tr>
<tr>
<td>2.3%</td>
<td>0.19</td>
<td>1</td>
<td>257.0</td>
<td>initmt_LOOP.4.11.177</td>
</tr>
<tr>
<td>2.3%</td>
<td>0.19</td>
<td>257</td>
<td>257.0</td>
<td>initmt_LOOP.5.11.178</td>
</tr>
<tr>
<td>2.2%</td>
<td>0.18</td>
<td>66,049</td>
<td>513.0</td>
<td>initmt_LOOP.6.11.179</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.00</td>
<td>--</td>
<td>--</td>
<td>initcomm</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.00</td>
<td>0</td>
<td>--</td>
<td>mpi_init(sync)</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.00</td>
<td>0</td>
<td>--</td>
<td>MPI_INIT</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.00</td>
<td>0</td>
<td>--</td>
<td>_STOP3</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.00</td>
<td>0</td>
<td>--</td>
<td>mpi_finalize(sync)</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.00</td>
<td>--</td>
<td>--</td>
<td>initmax</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.00</td>
<td>1</td>
<td>2.0</td>
<td>initmax_LOOP.5.11.374</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.00</td>
<td>1</td>
<td>2.0</td>
<td>initmax_LOOP.1.11.344</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.00</td>
<td>1</td>
<td>2.0</td>
<td>initmax_LOOP.3.11.359</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.00</td>
<td>1</td>
<td>2.0</td>
<td>initmax_LOOP.2.11.351</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.00</td>
<td>1</td>
<td>2.0</td>
<td>initmax_LOOP.4.11.366</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.00</td>
<td>1</td>
<td>2.0</td>
<td>initmax_LOOP.6.11.381</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.00</td>
<td>0</td>
<td>--</td>
<td>MP3_FINALIZE</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.00</td>
<td>0</td>
<td>--</td>
<td>call_init.part.0</td>
</tr>
</tbody>
</table>

Table 1: Calltree with Loop Inclusive Time
Crucial computation loops: compiler loopmark info

```fortran
SUBROUTINE jacobi(nn, gosa)

!********************************************************************
IMPLICIT REAL*4(a-h,o-z)
 INTEGER, INTENT(IN) :: nn
 REAL, INTENT(OUT) :: gosa
 INTEGER*4 istat
INCLUDE 'mpif.h'
INCLUDE 'param.h'

+ 1 2 3 4
> DO loop=1,nn
1       gosa=0.0
1       wgosa=0.0
1
1 + 1 2 3 4 < DO k=2,kmax-1
1 + 1 2 3 4 < DO j=2,jmax-1
1 2 3 4 < DO i=2,imax-1
1 2 3 4 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) &
1 2 3 4 +b(i,j,k,1)*p(i+1,j+1,k)-p(i+1,j-1,k)-p(i-1,j,k+1)+p(i-1,j,k-1)) &
1 2 3 4 +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)) &
1 2 3 4 +b(i,j,k,3)*p(i+1,j,k+1)-p(i-1,j,k+1)-p(i-1,j,k-1)+p(i-1,j,k-1)) &
1 2 3 4 +c(i,j,k,1)*p(i-1,j,k)+c(i,j,k,2)*p(i,j-1,k)) &
1 2 3 4 +c(i,j,k,3)*p(i,j,k-1)+wrk1(i,j,k)
1 2 3 4 ss=(s0*a(i,j,k,4)-p(i,j,k))"bnd(i,j,k)
1 2 3 4 wgosa=wgosa+ss"ss
1 2 3 4 wrk2(i,j,k)=p(i,j,k)+omega *ss
1 2 3 4 END DO
1 2 3 4 END DO
1 2 3 4 END DO
1 2 3 4 END DO

+ 1 2 3 4 < DO k=2,kmax-1
+ 1 2 3 4 < DO j=2,jmax-1
+ 1 2 3 4 < DO i=2,imax-1
    1 2 3 4 p(i,j,k)=wrk2(i,j,k)
1 2 3 4 END DO
1 2 3 4 END DO
1 2 3 4 END DO
```

© Copyright 2023 Hewlett Packard Enterprise Development LP
Let’s optimize that code

- Shared memory parallelism
- We have a tool called reveal to help parallelize code for host and GPU
- We need to build a **program library** to use reveal

```bash
module unload perftools-lite-loops
reveal himeno.pl himeno.exe+27714-8747979t
```

- The program library is needed so we can enable full inter-procedural analysis.
<table>
<thead>
<tr>
<th>Scope?</th>
<th>Line #</th>
<th>File or Source Line</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>156</td>
<td>Loop in function INITMT</td>
</tr>
<tr>
<td></td>
<td>157</td>
<td>Loop in function INITMT</td>
</tr>
<tr>
<td></td>
<td>158</td>
<td>Loop in function INITMT</td>
</tr>
<tr>
<td></td>
<td>177</td>
<td>Loop in function INITMT</td>
</tr>
<tr>
<td></td>
<td>178</td>
<td>Loop in function INITMT</td>
</tr>
<tr>
<td></td>
<td>179</td>
<td>Loop in function INITMT</td>
</tr>
<tr>
<td></td>
<td>216</td>
<td>Loop in function JACOBI</td>
</tr>
<tr>
<td></td>
<td>220</td>
<td>Loop in function JACOBI</td>
</tr>
<tr>
<td></td>
<td>221</td>
<td>Loop in function JACOBI</td>
</tr>
<tr>
<td></td>
<td>222</td>
<td>Loop in function JACOBI</td>
</tr>
<tr>
<td></td>
<td>236</td>
<td>Loop in function JACOBI</td>
</tr>
</tbody>
</table>

**List of Loops to be Scoped**

- Apply Filter
- Time: 0.000
- Trips: 2
- Threads: 4
- Speedup: 0.010

[Scope For CPU] [Scope For GPU] [Cancel]
### himeno.f90: Loop@220

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Scope</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Array</td>
<td>P</td>
<td>S</td>
</tr>
<tr>
<td>b</td>
<td>Array</td>
<td>P</td>
<td>S</td>
</tr>
<tr>
<td>bnd</td>
<td>Array</td>
<td>P</td>
<td>S</td>
</tr>
<tr>
<td>c</td>
<td>Array</td>
<td>P</td>
<td>S</td>
</tr>
<tr>
<td>i</td>
<td>Scalar</td>
<td>P</td>
<td>S</td>
</tr>
<tr>
<td>imax</td>
<td>Scalar</td>
<td>P</td>
<td>S</td>
</tr>
<tr>
<td>j</td>
<td>Scalar</td>
<td>P</td>
<td>S</td>
</tr>
<tr>
<td>jmax</td>
<td>Scalar</td>
<td>P</td>
<td>S</td>
</tr>
<tr>
<td>k</td>
<td>Scalar</td>
<td>P</td>
<td>S</td>
</tr>
<tr>
<td>kmax</td>
<td>Scalar</td>
<td>P</td>
<td>S</td>
</tr>
<tr>
<td>omega</td>
<td>Scalar</td>
<td>P</td>
<td>S</td>
</tr>
<tr>
<td>p</td>
<td>Array</td>
<td>P</td>
<td>S</td>
</tr>
<tr>
<td>s0</td>
<td>Scalar</td>
<td>P</td>
<td>S</td>
</tr>
<tr>
<td>ss</td>
<td>Scalar</td>
<td>P</td>
<td>S</td>
</tr>
<tr>
<td>wgosa</td>
<td>Scalar</td>
<td>R</td>
<td>S</td>
</tr>
<tr>
<td>wrk1</td>
<td>Array</td>
<td>P</td>
<td>S</td>
</tr>
<tr>
<td>wrk2</td>
<td>Array</td>
<td>P</td>
<td>S</td>
</tr>
</tbody>
</table>

**Options:**
- First/Last Private
- Enable FirstPrivate
- Enable LastPrivate
- Reduction:
  - None

**Find Name:** [Input Field]

**Insert Directive** | **Show Directive** | **Close**
DO k=2,kmax-1
  DO j=jmax-1,1,-1
    DO i=1,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,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)-p(i,j+1,k-1)
      &+c(i,j,k,1)*p(i+1,j+1,k)-p(i,j+1,k-1)
      &+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)
    wgosaw=wgosc+ss*ss
    wrk2(i,j,k)=p(i,j,k)+omega*ss
  END
  END
END
Threading performance improvement

- We fix the number of iterations to be 50, so that we can compare time to solutions for different versions
- Remember that the benchmark is memory-bandwidth limited
- Still OpenMP gave a small benefit
  - 8% with 2 threads with respect to 1 thread on LUMI
Now let’s put it on the GPU

- Go back to Reveal and generate GPU code
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Map</th>
<th>Extents</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>a G</td>
<td>Array</td>
<td>P</td>
<td>T F TF</td>
<td>A C U</td>
</tr>
<tr>
<td>b G</td>
<td>Array</td>
<td>P</td>
<td>T F TF</td>
<td>A C U</td>
</tr>
<tr>
<td>bnd G</td>
<td>Array</td>
<td>P</td>
<td>T F TF</td>
<td>A C U</td>
</tr>
<tr>
<td>c G</td>
<td>Array</td>
<td>P</td>
<td>T F TF</td>
<td>A C U</td>
</tr>
<tr>
<td>i G</td>
<td>Scalar</td>
<td>P</td>
<td>T F TF</td>
<td>A C U</td>
</tr>
<tr>
<td>jmax</td>
<td>Scalar</td>
<td>P</td>
<td>T F TF</td>
<td>A C U</td>
</tr>
<tr>
<td>j</td>
<td>Scalar</td>
<td>P</td>
<td>T F TF</td>
<td>A C U</td>
</tr>
<tr>
<td>jmax</td>
<td>Scalar</td>
<td>P</td>
<td>T F TF</td>
<td>A C U</td>
</tr>
<tr>
<td>k G</td>
<td>Scalar</td>
<td>P</td>
<td>T F TF</td>
<td>A C U</td>
</tr>
<tr>
<td>kmax</td>
<td>Scalar</td>
<td>P</td>
<td>T F TF</td>
<td>A C U</td>
</tr>
<tr>
<td>omega</td>
<td>Scalar</td>
<td>P</td>
<td>T F TF</td>
<td>A C U</td>
</tr>
<tr>
<td>p G</td>
<td>Array</td>
<td>P</td>
<td>T F TF</td>
<td>A C U</td>
</tr>
<tr>
<td>s0</td>
<td>Scalar</td>
<td>P</td>
<td>T F TF</td>
<td>A C U</td>
</tr>
<tr>
<td>ss</td>
<td>Scalar</td>
<td>P</td>
<td>T F TF</td>
<td>A C U</td>
</tr>
<tr>
<td>wgos</td>
<td>Scalar</td>
<td>P</td>
<td>T F TF</td>
<td>A C U</td>
</tr>
<tr>
<td>wrkl</td>
<td>Array</td>
<td>P</td>
<td>T F TF</td>
<td>A C U</td>
</tr>
<tr>
<td>wrkl2</td>
<td>Array</td>
<td>P</td>
<td>T F TF</td>
<td>A C U</td>
</tr>
</tbody>
</table>
Initial performance

- Performance obtained on LUMI (a single node):
  
<table>
<thead>
<tr>
<th>Performance</th>
<th>GFLOPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU only (8 ranks x 2 threads)</td>
<td>75</td>
</tr>
<tr>
<td>Initial GPU porting (8 ranks, 8 GPUs)</td>
<td>28</td>
</tr>
</tbody>
</table>

- Why is GPU execution performing worse?
  - Check data movement!

- Perftools-lite-gpu specifically target GPU profiling

% module load craype-accel-amd-gfx90a rocm
% module load perftools-lite-gpu
% 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. Time%</th>
<th>Calls</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PE=HIDE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Thread=HIDE</td>
</tr>
<tr>
<td>100.0%</td>
<td>22.400068</td>
<td>--</td>
<td>--</td>
<td>22,546.6</td>
<td>Total</td>
</tr>
<tr>
<td>86.5%</td>
<td>19.372358</td>
<td>--</td>
<td>--</td>
<td>20,617.0</td>
<td>OACC</td>
</tr>
<tr>
<td>77.1%</td>
<td>17.266184</td>
<td>0.836747</td>
<td>5.3%</td>
<td>3,904.0</td>
<td>cray_acc_transfer_list</td>
</tr>
<tr>
<td>9.4%</td>
<td>2.099139</td>
<td>0.030002</td>
<td>1.6%</td>
<td>16,473.0</td>
<td>acc_get_device_num</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.005400</td>
<td>0.001280</td>
<td>21.9%</td>
<td>60.0</td>
<td>jacobi_.ACC_COPY@li.221</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000964</td>
<td>0.000331</td>
<td>29.2%</td>
<td>60.0</td>
<td>jacobi_.ACC_COPY@li.243</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000388</td>
<td>0.000073</td>
<td>18.0%</td>
<td>60.0</td>
<td>jacobi_.ACC_ASYNC_KERNEL@li.221</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000284</td>
<td>0.000041</td>
<td>14.4%</td>
<td>60.0</td>
<td>jacobi_.ACC_SYNC_WAIT@li.243</td>
</tr>
<tr>
<td>7.4%</td>
<td>1.665387</td>
<td>--</td>
<td>--</td>
<td>63.0</td>
<td>USER</td>
</tr>
<tr>
<td>4.5%</td>
<td>1.005416</td>
<td>0.149915</td>
<td>14.8%</td>
<td>2.0</td>
<td>jacobi_</td>
</tr>
<tr>
<td>2.9%</td>
<td>0.659721</td>
<td>0.007622</td>
<td>1.3%</td>
<td>1.0</td>
<td>himenobmtxp_</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000249</td>
<td>0.000058</td>
<td>21.5%</td>
<td>60.0</td>
<td>jacobi_.ACC_REGION@li.221</td>
</tr>
<tr>
<td>3.7%</td>
<td>0.819459</td>
<td>--</td>
<td>--</td>
<td>983.0</td>
<td>MPI</td>
</tr>
</tbody>
</table>

...
### Perftools-lite-gpu profile

#### Table 9: Time and Bytes Transferred for Accelerator Regions

<table>
<thead>
<tr>
<th>Time%</th>
<th>Time%</th>
<th>Acc Time</th>
<th>Acc Time</th>
<th>Acc Copy In (MiBytes)</th>
<th>Acc Copy Out (MiBytes)</th>
<th>Events</th>
<th>Calltree</th>
<th>Accelerator ID PE=HIDE</th>
<th>Thread=HIDE</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td>22.400068</td>
<td>100.0%</td>
<td>17.23</td>
<td>221,399</td>
<td>15,814</td>
<td>3,725</td>
<td>Total</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0%</td>
<td>22.400035</td>
<td>100.0%</td>
<td>17.23</td>
<td>221,399</td>
<td>15,814</td>
<td>3,725</td>
<td>himenobmtxp_</td>
<td></td>
<td></td>
</tr>
<tr>
<td>95.8%</td>
<td>21.467652</td>
<td>100.0%</td>
<td>17.23</td>
<td>221,399</td>
<td>15,814</td>
<td>3,725</td>
<td>jacobi_</td>
<td></td>
<td></td>
</tr>
<tr>
<td>78.3%</td>
<td>17.530224</td>
<td>100.0%</td>
<td>17.23</td>
<td>221,399</td>
<td>15,814</td>
<td>3,724</td>
<td>jacobi_.ACC_REGION@li.221</td>
<td></td>
<td></td>
</tr>
<tr>
<td>72.2%</td>
<td>16.180262</td>
<td>92.2%</td>
<td>15.89</td>
<td>221,399</td>
<td>--</td>
<td>2,484</td>
<td>jacobi_.ACC_COPY@li.221</td>
<td></td>
<td></td>
</tr>
<tr>
<td>72.2%</td>
<td>16.174863</td>
<td>--</td>
<td>--</td>
<td>110,700</td>
<td>--</td>
<td>2,424</td>
<td>acc.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0%</td>
<td>0.005400</td>
<td>92.2%</td>
<td>15.89</td>
<td>110,700</td>
<td>--</td>
<td>7,907</td>
<td>acc.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>60</td>
<td>jacobi_.ACC_COPY@li.221(exclusive)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>60</td>
<td>jacobi_.ACC_COPY@li.221(exclusive)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>acc.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>acc.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>acc.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>acc.0</td>
</tr>
</tbody>
</table>
Compiler loopmark information

216.  + 1------------------< DO loop=1,nn
217.     1                 gosa=0.0
218.     1                 wgosa=0.0
219.     1
220.     1                 ! Directive inserted by Cray Reveal. May be incomplete.
221.  + 1 MG----------------< !$OMP target teams distribute &
222.     1 MG             !$OMP& private(i, j, k, s0, ss) &
223.     1 MG             !$OMP& firstprivate(imax, jmax, kmax, omega) &
224.     1 MG             !$OMP& map( tofrom: wgosa ) &
225.     1 MG             !$OMP& reduction(+: wgosa) &
226.     1 MG             !$OMP& map(always, tofrom: wgosa) &
227.     1 MG             !$OMP& reduction(+: wgosa) &
228.     1 MG             !$OMP& map(always, tofrom: wrk2(:,:,:)) &
229.     1 MG             !$OMP& map(always, tofrom: wrk2(:,:,:)) &
230.     1 MG             !$OMP& map(always, tofrom: wrk2(:,:,:)) &
231.     1 MG             !$OMP& map(always, tofrom: wrk2(:,:,:)) &
232.     1 MG             !$OMP& map(always, tofrom: wrk2(:,:,:)) &
233.     1 MG             !$OMP& map(always, tofrom: wrk2(:,:,:)) &
234.     1 MG             !$OMP& map(always, tofrom: wrk2(:,:,:)) &
235.     1 MG             !$OMP& map(always, tofrom: wrk2(:,:,:)) &
236.     1 MG             !$OMP& map(always, tofrom: wrk2(:,:,:)) &
237.     1 MG             !$OMP& map(always, tofrom: wrk2(:,:,:)) &
238.     1 MG             !$OMP& map(always, tofrom: wrk2(:,:,:)) &
239.     1 MG             !$OMP& map(always, tofrom: wrk2(:,:,:)) &
240.     1 MG             !$OMP& map(always, tofrom: wrk2(:,:,:)) &
241.     1 MG             !$OMP& map(always, tofrom: wrk2(:,:,:)) &
242.     1 MG             !$OMP& map(always, tofrom: wrk2(:,:,:)) &
243.     1 MG             !$OMP& map(always, tofrom: wrk2(:,:,:)) &
Complications with Himeno

- All Arrays are in COMMON blocks – these are globally available, so we need a way to keep them on the GPU all the time.

- 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 slides outline some optimizations for himeno
  - 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
Allocate the data on the GPU once: data environment

\[ \begin{align*}
      \text{my} &= \text{my0} - 1 \\
      \text{mz} &= \text{mz0} - 1 \\
      \end{align*} \]

!$\text{OMP TARGET DATA MAP (alloc:a,b,bnd,c,p,wrk1,wrk2)}$

!C Initializing communicator
CALL initcomm

!C Initializing computational index
CALL initmax(mx,my,mz,it)

... 

\[ \begin{align*}
      \text{gosa} &= 0.0 \\
      \text{cpu} &= 0.0 \\
      \text{CALL mpi\_barrier(mpi\_comm\_world,ierr)} \\
      \text{cpu0} &= \text{mpi\_wtime()} \\
      \text{! Jacobi Iteration} \\
      \text{CALL jacobi(nn,gosa)} \\
      \text{cpu1} &= \text{mpi\_wtime()} - \text{cpu0} \\
      \end{align*} \]

!$\text{OMP END TARGET DATA}$
Initialize arrays on the accelerator

156. + MG----------------------< !$OMP target teams distribute &
157. MG $OMP& collapse(3) &
158. MG $OMP& private(i, j, k)
159. MG C---------------------< DO k=1,mkmax
160. MG C C-------------------< DO j=1,mjmax
161. MG C C C---------------< DO i=1,mimax
162. MG C C gCr2           a(i,j,k,1)=0.0
163. MG C C gCr2           a(i,j,k,2)=0.0
164. MG C C gCr2           a(i,j,k,3)=0.0
165. MG C C gCr2           a(i,j,k,4)=0.0
166. MG C C gCr2           b(i,j,k,1)=0.0
167. MG C C gCr2           b(i,j,k,2)=0.0
168. MG C C gCr2           b(i,j,k,3)=0.0
169. MG C C gCr2           c(i,j,k,1)=0.0
170. MG C C gCr2           c(i,j,k,2)=0.0
171. MG C C gCr2           c(i,j,k,3)=0.0
172. MG C C gCr2 A--------< p(i,j,k)=0.0
173. MG C C gCr2 A--------< wrk1(i,j,k)=0.0
174. MG C C gCr2 A--------< wrk2(i,j,k)=0.0
175. MG C C gCr2 A--------< bnd(i,j,k)=0.0
176. MG C C gCr2------------ END DO
177. MG C C---------------- END DO
178. MG C------------------- END DO
Initialize arrays on the accelerator

180. + MG--------------< !$OMP target teams distribute
181. MG !$OMP& collapse(3) &
182. MG !$OMP& private(i, j, k) &
183. MG C--------------< DO k=1,kmax
184. MG C C------------< DO j=1,jmax
185. + MG C C gCr8------< DO i=1,imax
186. MG C C gCr8 a(i,j,k,1)=1.0
187. MG C C gCr8 a(i,j,k,2)=1.0
188. MG C C gCr8 a(i,j,k,3)=1.0
189. MG C C gCr8 a(i,j,k,4)=1.0/6.0
190. MG C C gCr8 b(i,j,k,1)=0.0
191. MG C C gCr8 b(i,j,k,2)=0.0
192. MG C C gCr8 b(i,j,k,3)=0.0
193. MG C C gCr8 c(i,j,k,1)=1.0
194. MG C C gCr8 c(i,j,k,2)=1.0
195. MG C C gCr8 c(i,j,k,3)=1.0
196. MG C C gCr8 p(i,j,k)=FLOAT((k-1+it)*(k-1+it)) /FLOAT((mz-1)*(mz-1))
197. MG C C gCr8 wrk1(i,j,k)=0.0
198. MG C C gCr8 wrk2(i,j,k)=0.0
199. MG C C gCr8 bnd(i,j,k)=1.0
200. MG C C gCr8------> END DO
201. MG C C------------> END DO
202. MG C---------------> END DO
Use these arrays in the main iterations

```c
 !$OMP  target teams distribute &
 !$OMP& private(i, j, k, s0, ss) &
 !$OMP& map(tofrom: wgosa) MAP(present,to:a,b,c,wrk1,wrk2,bnd) &
 !$OMP& reduction(+: wgosa)
 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,k-1)) -p(i,j+1,k)+p(i,j-1,k-1)) &
         +b(i,j,k,3)*(p(i+1,j,k+1)-p(i-1,j,k+1)) -p(i+1,j,k-1)+p(i-1,j,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
     END DO
   END DO
 END DO
```

Saying that these arrays are already mapped to the GPU
Swap arrays and update halos

```fortran
!$OMP target teams distribute &
!$OMP& private(i, j, k)
DO k=2,kmax-1
  DO j=2,jmax-1
    DO i=2,imax-1
      p(i,j,k)=wrk2(i,j,k)
    END DO
  END DO
END DO

!$OMP TARGET UPDATE from(p)
CALL sendp(ndx,ndy,ndz)

CALL mpi_allreduce(wgosa, gosa, &
  1, mpi_real4, &
  mpi_sum, mpi_comm_world, &
  ierr)

END DO
! End of iteration
```

Need to get p back to host for MPI
Performance with data regions

- Performance obtained on LUMI (a single node):

<table>
<thead>
<tr>
<th>Description</th>
<th>GFLOPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU only (8 ranks x 2 threads)</td>
<td>75</td>
</tr>
<tr>
<td>Initial GPU porting (8 ranks, 8 GPUs)</td>
<td>28</td>
</tr>
<tr>
<td>Data transfer optimized GPU porting (8 ranks, 8 GPUs)</td>
<td>368</td>
</tr>
</tbody>
</table>
Let’s deal with the Halo exchanges

- There are some options

- We want to avoid copying the P array to do halo swaps
  - We can copy the halos to contiguous buffers on the GPU and only move those between GPU and CPU

- Next, we can consider directly communicating the halo data from GPU to GPU
  - This is supported in Cray MPICH if we pass GPU pointers to MPI

- Let’s start with the buffer copying approach...
Halo updates

```c
!$OMP TARGET DATA MAP (alloc: xr_s, xr_n, xs_s, xs_n)
call mpi_irecv(XR_S(1),
      > mjmax*mkmax,
      > MPI_real,
      > npx(1),
      > 2,
      > mpi_comm_world,
      > ireq(3),
      > ierr)
call mpi_irecv(XR_N(1),
      > mjmax*mkmax,
      > MPI_real,
      > npx(2),
      > 1,
      > mpi_comm_world,
      > ireq(2),
      > ierr)

!$OMP target teams distribute map(from: xs_s, xs_n) private(icnt)
DO K = 1, mkmax
  DO J = 1, mjmax
    ICNT = (K-1)*mjmax+J
    XS_S(ICNT)=P(2,J,K)
    XS_N(ICNT)=P(mimax-1,J,K)
  END DO
END DO
!$OMP TARGET UPDATE from(xs_s, xs_n)
call mpi_isend(XS_S(1),
      > mjmax*mkmax,
      > MPI_real,
      > npx(1),
      > 1,
```

Buffers on GPU
Halo updates

call mpi_waitall(4,
   > ireq,
   > ist,
   > ierr)

c !$OMP TARGET UPDATE to(xr_s,xr_n)
 !$OMP target teams distribute map(from:xr_s,xr_n) private(icnt)
  DO K = 1,mkmax
   DO J = 1,mjmax
      ICNT = (K-1)*mjmax+J
      if(npx(1).ne.-1)P(mimax,J,K) = XR_S(ICNT)
      if(npx(2).ne.-1)P(1,J,K) = XR_N(ICNT)
   END DO
  END DO
 !$OMP END TARGET DATA

return
end
Table 1: Profile by Function Group and Function

<table>
<thead>
<tr>
<th>Time%</th>
<th>Time</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Calls</th>
<th>Group</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>Time%</td>
<td></td>
<td></td>
<td>PE=HIDE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Thread=HIDE</td>
</tr>
</tbody>
</table>

---

| 100.0% | 0.967625 | -- | -- | 31,655.0 | Total |

| 65.9% | 0.637231 | -- | -- | 20,137.0 | OACC |

---

| 18.1% | 0.174761 | 0.001591 | 1.2% | 1.0 | initmt_.ACC_COPY@li.152 |
| 17.6% | 0.170772 | 0.000987 | 0.8% | 500.0 | jacobiat_.ACC_SYNC_WAIT@li.240 |
| 4.8%  | 0.046532 | 0.000311 | 0.9% | 500.0 | jacobiat_.ACC_SYNC_WAIT@li.250 |
| 3.3%  | 0.032382 | 0.000266 | 1.1% | 14.0 | jacobiat_.ACC_COPY@li.87 |
| 2.7%  | 0.026012 | 0.000636 | 3.2% | 503.0 | sendp1_.ACC_COPY@li.679 |
| 2.7%  | 0.025881 | 0.000367 | 1.9% | 503.0 | sendp2_.ACC_COPY@li.623 |
| 2.7%  | 0.025762 | 0.000490 | 2.5% | 503.0 | sendp2_.ACC_COPY@li.703 |
| 2.6%  | 0.024681 | 0.000240 | 1.3% | 503.0 | sendp2_.ACC_COPY@li.598 |
| 1.1%  | 0.010461 | 0.000029 | 0.4% | 1.0 | jacobiat_.ACC_COPY@li.212 |
| 1.0%  | 0.010089 | 0.000175 | 2.3% | 500.0 | jacobiat_.ACC_COPY@li.240 |
| 0.9%  | 0.008922 | 0.000933 | 1.4% | 500.0 | jacobiat_.ACC_COPY@li.217 |
| 0.8%  | 0.007708 | 0.000420 | 6.9% | 503.0 | sendp1_.ACC_SYNC_WAIT@li.678 |
| 0.8%  | 0.007384 | 0.000228 | 4.6% | 503.0 | sendp1_.ACC_COPY@li.784 |
| 0.8%  | 0.007291 | 0.000140 | 2.5% | 503.0 | sendp2_.ACC_COPY@li.598 |
| 0.5%  | 0.005304 | 0.000173 | 4.2% | 503.0 | sendp1_.ACC_ASYNC_KERNEL@li.704 |
| 0.5%  | 0.005155 | 0.000106 | 2.7% | 503.0 | sendp2_.ACC_ASYNC_KERNEL@li.624 |
| 0.5%  | 0.004888 | 0.000037 | 1.6% | 500.0 | jacobiat_.ACC_ASYNC_KERNEL@li.217 |
| 0.5%  | 0.004387 | 0.000250 | 7.2% | 503.0 | sendp1_.ACC_ASYNC_KERNEL@li.671 |
| 0.4%  | 0.004315 | 0.000104 | 3.1% | 500.0 | jacobiat_.ACC_ASYNC_KERNEL@li.242 |
| 0.4%  | 0.004123 | 0.000047 | 1.5% | 503.0 | sendp1_.ACC_ASYNC_KERNEL@li.711 |
| 0.4%  | 0.003967 | 0.000163 | 5.2% | 503.0 | sendp2_.ACC_ASYNC_KERNEL@li.608 |
| 0.4%  | 0.003436 | 0.000138 | 5.1% | 503.0 | sendp2_.ACC_SYNC_WAIT@li.597 |
Enabling GPU to GPU communication with MPI

- We can tell MPI to use GPU data
- We set MPICH_GPU_SUPPORT_ENABLED=1 for Cray MPICH
- We use the OpenMP `use_device_ptr` to specify we are using the variable on the GPU

- We can do this in the `sendp<n>` routines:

```c
!$OMP TARGET DATA MAP(alloc:zr_e,zr_w,zs_e,zs_w)
!$OMP TARGET DATA use_device_ptr(zr_e,zr_w,zs_e,zs_w)
   call mpi_irecv(ZR_E(1),
                   >        imax*jmax,
                   >        MPI_real,
```

Replace the “target update” with the GPU device pointer, so that MPI will run the GPU-aware MPI
Enabling GPU to GPU communication with MPI

- It turns out that Cray MPICH can accept GPU pointers when communicating MPI datatypes.
- So we could have just made our MPI calls and passed in the device pointer as follows:

```fortran
!$omp target data use_device_ptr(p)
! Call MPI calls for halo exchanges
CALL sendp(ndx,ndy,ndz)
!$omp end target data
```

- This does work with HPE Cray MPICH but MPI is not doing the packing on the GPU.

Replace the “target update” with the GPU device pointer, so that MPI will run the GPU-aware MPI.
Performance with data regions and G2G MPI on p

- Performance obtained on LUMI (a single node):

<table>
<thead>
<tr>
<th>Description</th>
<th>GFLOPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU only (8 ranks x 2 threads)</td>
<td>75 GFLOPs</td>
</tr>
<tr>
<td>Initial GPU porting (8 ranks, 8 GPUs)</td>
<td>28 GFLOPs</td>
</tr>
<tr>
<td>Data transfer optimized GPU porting (8 ranks, 8 GPUs)</td>
<td>368 GFLOPs</td>
</tr>
<tr>
<td>Data transfer optimized GPU porting with G2G (8 ranks, 8 GPUs)</td>
<td>374 GFLOPs</td>
</tr>
</tbody>
</table>
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
export 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
export 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)
export CRAY_ACC_DEBUG=3

ACC: Start transfer 29 items from src/fluxi.f:21
ACC:   flags: RETURN_ACC_TIME
ACC:
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
Output from the initial GPU version (Jacobi iterations)

0: ACC: Version 5.0 of HIP already initialized, runtime version 50221153
0: ACC: Get Device 0
0: ACC: Set Thread Context
0: ACC: Start transfer 10 items from himeno.f90:221
0: ACC: allocate, copy to acc 'a(:,,:,:)' (552747440 bytes)
0: ACC: allocate, copy to acc 'b(:,,:,:)' (414560580 bytes)
0: ACC: allocate, copy to acc 'bnd(:,,:,:)' (138186860 bytes)
0: ACC: allocate, copy to acc 'c(:,,:,:)' (414560580 bytes)
0: ACC: allocate, copy to acc 'p(:,,:,:)' (138186860 bytes)
0: ACC: allocate, copy to acc 'wgosa' (4 bytes)
0: ACC: allocate, copy to acc 'wrk1(:,,:,:)' (138186860 bytes)
0: ACC: allocate, copy to acc 'wrk2(:,,:,:)' (138186860 bytes)
0: ACC: allocate reusable, copy to acc <internal> (4 bytes)
0: ACC: allocate reusable <internal> (1020 bytes)
0: ACC: End transfer (to acc 1934616048 bytes, to host 0 bytes)
0: ACC: Execute kernel jacobi_$ck_L221_1 blocks:255 threads:256 async(auto) from himeno.f90:221
0: ACC: Wait async(auto) from himeno.f90:243
0: ACC: Start transfer 10 items from himeno.f90:243
0: ACC: free 'a(:,,:,:)' (552747440 bytes)
0: ACC: free 'b(:,,:,:)' (414560580 bytes)
0: ACC: free 'bnd(:,,:,:)' (138186860 bytes)
0: ACC: free 'c(:,,:,:)' (414560580 bytes)
0: ACC: free 'p(:,,:,:)' (138186860 bytes)
0: ACC: copy to host, free 'wgosa' (4 bytes)
0: ACC: free 'wrk1(:,,:,:)' (138186860 bytes)
0: ACC: copy to host, free 'wrk2(:,,:,:)' (138186860 bytes)
0: ACC: done reusable <internal> (4 bytes)
0: ACC: done reusable <internal> (1020 bytes)
0: ACC: End transfer (to acc 0 bytes, to host 138186864 bytes)

Repeated per each Jacobi iteration

We are moving a lot of data
Output from the GPU version with data movement optimized (initialization)

0: ACC: Version 5.0 of HIP already initialized, runtime version 50221153
0: ACC: Get Device 0
0: ACC: Set Thread Context
0: ACC: Start transfer 7 items from himeno.f90:58
  0: ACC: allocate 'a(:,:)' (552747440 bytes)
  0: ACC: allocate 'b(:,:)' (414560580 bytes)
  0: ACC: allocate 'bnd(:,:)' (138186860 bytes)
  0: ACC: allocate 'o(:,:)' (414560580 bytes)
  0: ACC: allocate 'p(:,:)' (138186860 bytes)
  0: ACC: allocate 'wrk1(:,:)' (138186860 bytes)
  0: ACC: allocate 'wrk2(:,:)' (138186860 bytes)
0: ACC: End transfer (to acc 0 bytes, to host 0 bytes)
0: ACC: Start transfer 7 items from himeno.f90:67
  0: ACC: present 'a(:,:)' (552747440 bytes)
  0: ACC: present 'b(:,:)' (414560580 bytes)
  0: ACC: present 'bnd(:,:)' (138186860 bytes)
  0: ACC: present 'o(:,:)' (414560580 bytes)
  0: ACC: present 'p(:,:)' (138186860 bytes)
  0: ACC: present 'wrk1(:,:)' (138186860 bytes)
  0: ACC: present 'wrk2(:,:)' (138186860 bytes)
0: ACC: End transfer (to acc 0 bytes, to host 0 bytes)
0: ACC: Execute kernel himenobmtxp_$ck_L67_1 blocks:134949 threads:256 async(auto) from himeno.f90:67
0: ACC: Wait async(auto) from himeno.f90:67
Output from GPU version with G2G (halo swaps)

0: ACC: Start transfer 1 items from himeno.f90:261
0: ACC: present 'p(:,:,;)' (138186860 bytes)
0: ACC: End transfer (to acc 0 bytes, to host 0 bytes)
0: ACC: Start transfer 1 items from himeno.f90:264
0: ACC: release present 'p(:,:,;)' (138186860 bytes)
0: ACC: End transfer (to acc 0 bytes, to host 0 bytes)

261 !$omp target data use_device_ptr(p)
262 ! Call MPI calls for halo exchanges
263 CALL sendp(ndx,ndy,ndz)
264 !$omp end target data
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
  - Target the time-consuming parts of the application
  - Be concerned with data movement (keep data on GPU)
  - Only move data when needed
- GPU direct is best way to do the message passing – in this case it only matters at scale