EuroHack17 @CSCS

4-8 september 2017

Lugano

TEAM ALYA (BSC)

Herbert Owen <herbert.owen@bsc.es>
Ricard Borrell <ricard.borrell@bsc.es>
Vishal Mehta <vishal.mehta@bsc.es>
Alistair Hart <ahart@cray.com>
Manuel Arenaz <manuel.arenaz@appentra.com>

GPU HACKATHON

INITIAL PROFILE

Its essentially a Linear Algebra problem.

We were already solving Ax=b on GPU

We wanted to assemble Ax=b as well on GPU.

Challenges

Lot of different types of physics & people involved in assembly of Ax=b.

Easy GPU use for scientists.

Evolution and Strategy

Refactor the code to be used for vectorization as well as GPUs.

Use OpenACC for implementing the assembly.

Conceptual understanding of OpenAcc and the learning the right approach to GPUs.

GPU HACKATHON

Problems Encountered

- Problems with legacy app structure -- Yes some refactoring needed
- Issues with algorithm -- NO
- Tool bugs -- Yes, cray wrapper and profile
- Tool lack of features -- NO
- System setup --- NO

Performance baseline

#CPU cores	VECTOR SIZE 8
12	6.21
24	2.99

- Workloads: **SMALL (2M elements)**
- Systems: **CSCS**(Piz Daint)
- Compilers: **INTEL**
- Paradigms: **MPI**
- svn version: **7331**

#CPU cores	VECTOR SIZE 8
12	10.30
24	4.90

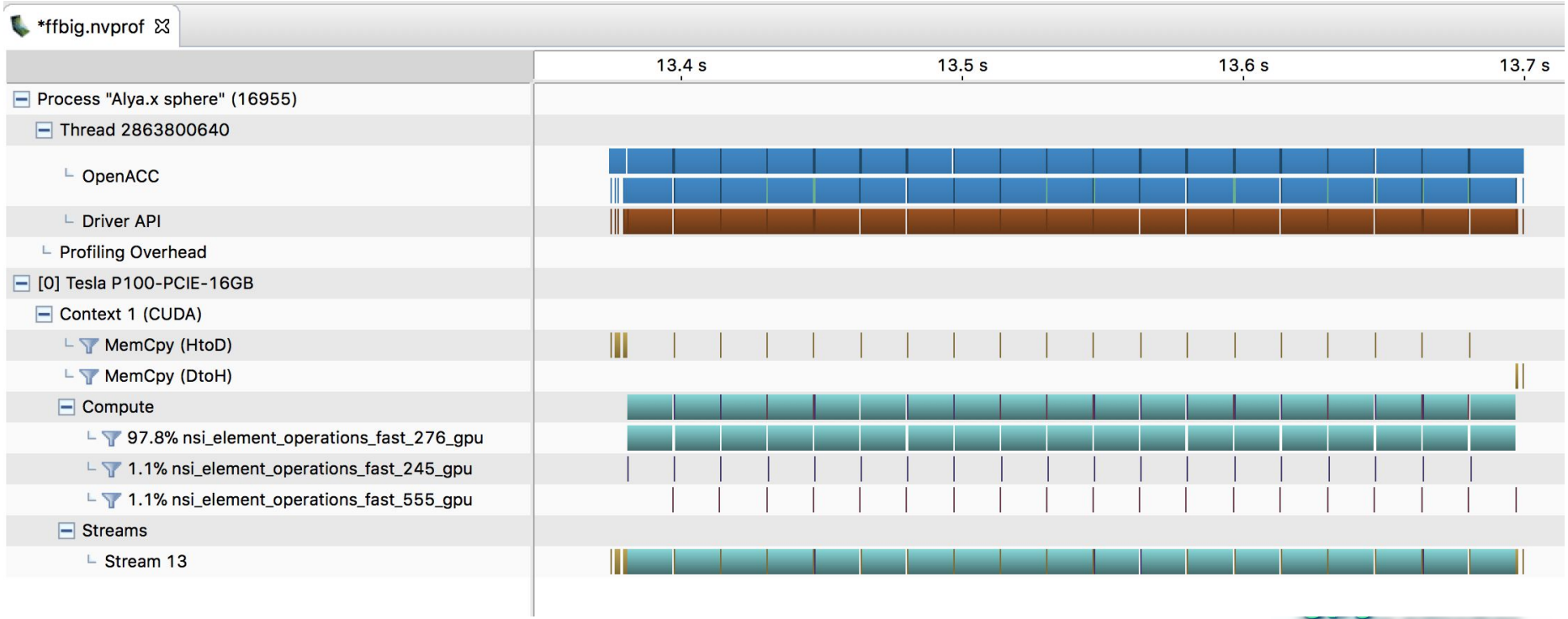
- Workloads: **SMALL (2M elements)**
- Systems: **CSCS**(Piz Daint)
- Compilers: **PGI**
- Paradigms: **MPI**
- svn version: **7331**

Performance on GPUs

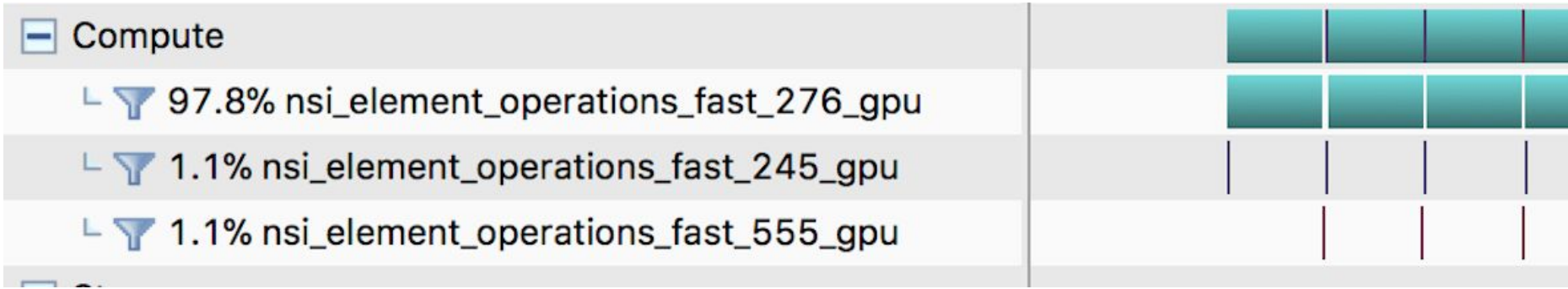
2GPU compared to 24mpi

VECTOR_SIZE	EXECUTION TIME ON 2 GPUs
16k	2.6
32K	2.01
64K	1.88
128K	**1.64**(2x to pizdaint cpu)
256K	1.65

PROFILING



PROFILING



SUMMARY OF THINGS DONE DURING EUROHACK 17

- !$acc parallel loop gang vector default(present) DONE OVER A BIG GROUP OF ELEMENTS
- Add copy in/out & create - For module variables Pragma “enter data” and “exit data”
- Some optimizations to the code: reduce redundant code & registers.

GPU HACKATHON

CONCLUSIONS

- Accomplished? OPENACC - 1SUB - 60% of time
- Did you achieve speed up? 2.4 against 1 PIZ-DAINT NODE . Also Works with MPI
- Create a new algorithm? NO
- Develop a new algorithm? NO
- Achieved new scientific goals? NO? Just faster
- Broke ground in your field (could this be done before?). NO. But before it seemed very complicated. Now it is easy.

GPU HACKATHON

Wishlist

- Slightly generalize kernel. Not just tetra
- Test an implicit case - matrix assembly.
- Use in other physics apart CFD, temperature, solids, etc.

GPU HACKATHON

Was it worth ??

- I first got in contact with OPENACC 1 month ago.
- I am a FE - CFD guy no experience with GPUs.
- I did not do most of the work myself but I have been able to follow nearly everything that has been done (Perhaps understand better register reduction).
- I can now do it for the other problems in Alya.

GPU HACKATHON

THANKS!

It was one of the best work trips I have done

Day 2: Start playing with openACC

#nodes/gpus	#cpus/node	VECTOR SIZE	TIME
2/0	**4**	**4**	**30.66**
2/2	4	1024	48.07
2/2	4	2048	31.71
2/2	**4**	**4096**	**28.45**
2/2	4	8192	33.27
2/2	**12**	**4096**	**29.16**
2/2	12	2048	33.09

- Workloads: **SMALL (2M elements)**
- Systems: **CSCS**(Piz Daint)
- Compilers: **PGI**
- Paradigms: **MPI+OpenACC**
- svn version: **7331**
- **Naïve approach only:** !$acc parallel loop

Other ideas

- Disable the colouring algorithm (it was coded to group sets of finite elements that don’t share mesh nodes and avoid race conditions):
  - Goal: Focus on a naive implementation of finite-elements assembly
  - Pattern: Parallel Sparse Reduction
  - Do benchmarking of a version that uses the OpenACC pragma “atomic” to prevent race conditions
- Analyze arithmetic-intensity of the code using Intel Advisor.
- Insights into compiler optimizations:
  - PGI (“-Minfo,accel,loop”)
  - CRAY (“-hlist=a” outputs a info file .lst)
  - GCC (??)
- Clause “collapse” to merge iteration spaces of perfectly nested loops
- Do profiling to identify the hotspots
  - Gprof, nvprof, craypat, others?

Day 1

- Create a GPU-oriented version with OpenACC: Focus on correctness
  - #pragma acc parallel loop
  - Fusion of loops
  - Scalarization of arrays 1D
- Technical issues:
  - PGI compiler automatically decides CPU-GPU transfers => SIGSEV
    - Enable HW dynamic memory (-managed) => OK

```
For (i=0; i<N; i++)
  A[i] = …

For (i=0; i<N; i++)
  B[i] = A[i]+1
```

```
For (i=0; i<N; i++)
  A[i] = …
  B[i] = A[i]+1
```

```
For (i=0; i<N; i++)
  T = …
  B[i] = T+1
```

GPU HACKATHON

Day 2

- Create a baseline execution to measure progress in performance:
  - Workloads: **SMALL (2M elements)**, MEDIUM
  - Paradigms: **MPI**, MPI+OpenACC
  - Create first optimized implementation using OpenACC:
  - GPU kernel structure:
    - Gather (global sparse matrix to local arrays)
    - Compute (dense computations on local arrays)
    - Scatter (local results to global sparse matrix
  - Need to protect the sparse accesses during “scatter” to prevent race conditions (using “#pragma acc atomic”):
    - Colouring algorithm group some finite elements may have a bug.
    - Atomic: No significant impact on performance :-))

GPU HACKATHON

Day 3

- Discuss performance-portability of OpenACC:
  - PGI - CRAY not compatible (2.0 vs 2.5) - will continue with PGI.
  - Removed “-managed”
  - Now use copyin, copyout & create (Beware update 2.5) for variables inside the subroutine
  - Pragma “enter data” and “exit data” to manage CPU-GPU data transfers for global module variables