Parallelizing Bioinformatics and Medical Imaging Applications: A Directive Based Approach

Collin Clark, Sunita Chandrasekaran

Objectives

The goal of this project was to update several applications from parallel implementation OpenACC standard 1.0 to 2.5 and compare the new multicore capabilities against that of Open Multi-Processor (OpenMP) 4.5

Particle Filter

Statistically estimates locations of target objects based on measurements of its original location and an estimated path. The specific implementation used is optimized for tracking leukocytes and myocardial cells [3].

Needleman-Wunch

Optimization method for DNA sequence alignments. Creates a 2D array of potential pairs from top left to bottom right and traverses the grid step-by-step to find the optimal alignment [2].

OpenACC

Directive-based parallel implementation used to offload programs in C, C++, and Fortran from a host CPU to an accelerator (GPUs, Power8, ARM processors, hybrid architectures, or other CPU cores) [4].

OpenMP

Directive-based parallel implementation used to easily convert single threaded applications to multicore ones [5].

Hardware

All measurements were taking on one of Nvidia's public access clusters on hsw_k80 nodes, equipped with dual Intel Xeon E5-2690 CPUs (20 cores per CPU, 3.00 GHz) and four Nvidia Tesla K80 cards (2 GPUs per card)

Software

Applications were compiled with The Portland Group, Inc. (PGI) compilers version 16.5 for the C and C++ programming languages on nodes running CentOS 7.2.1511 with Linux kernel 3.10.0-229-.14.1.el7.x86_64

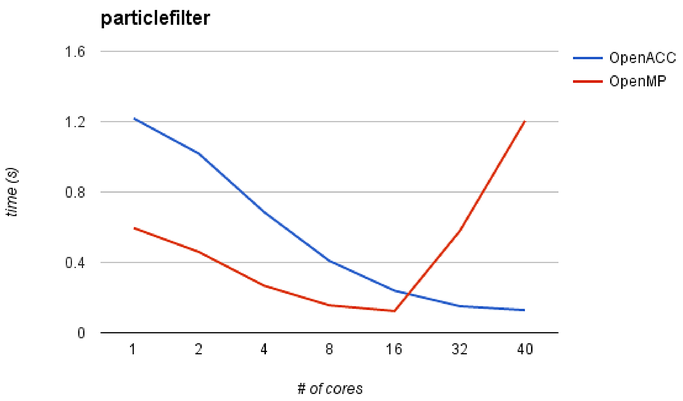


Figure 1: Particlefilter scaling with 128 x 128 x 10 grid dimensions

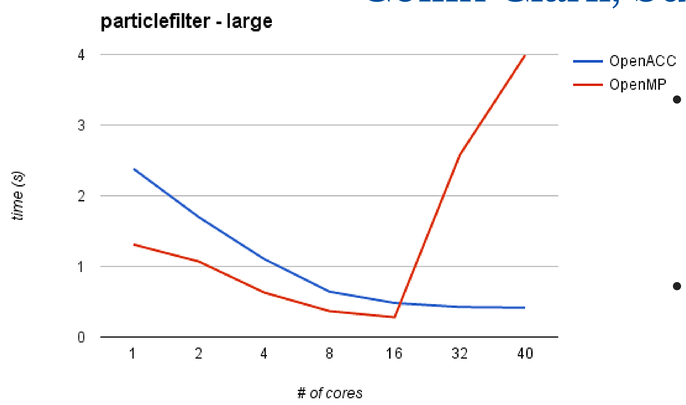


Figure 2: Particlefilter scaling with 256 x 256 x 20 grid dimensions

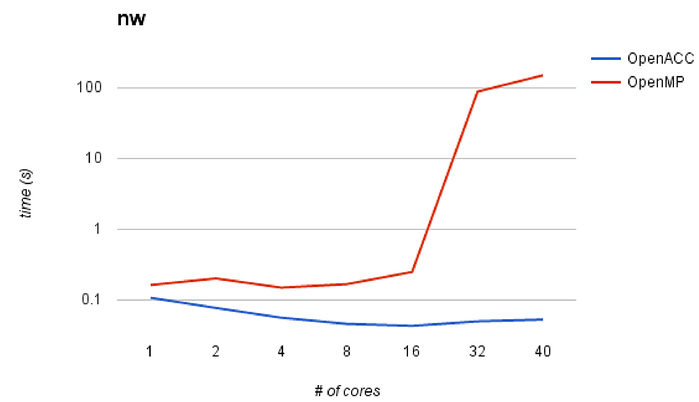


Figure 3: Needleman-Wunch scaling with 2048 x 2048 grid dimensions

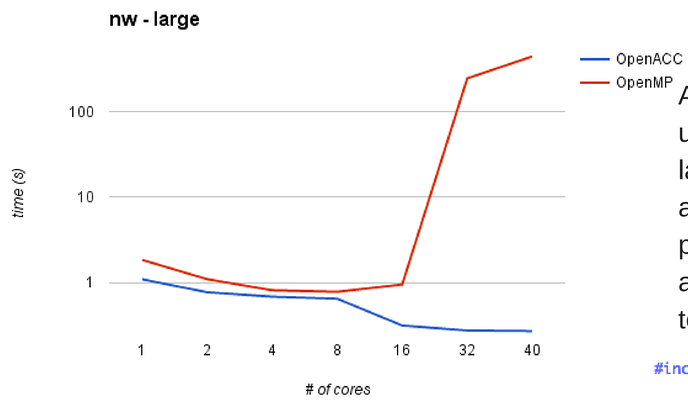


Figure 4: Needleman-Wunch scaling with 6144 x 6144 grid dimensions

Fig. 1-4

- The graphs depict two applications from the Rodinia Benchmark Suite [1], Particlefilter and Needleman-Wunch, and the execution times of each as the number of CPU cores increases.
- Each application has two datasets: one of the recommended size (leftmost) and one approximately 8x in size (rightmost).
- Y-axis is scaled on a per-chart basis to best display results

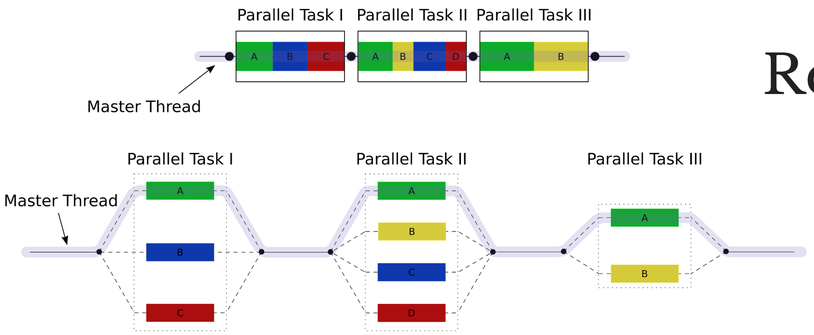
Data Collection

All of the timing data was measured using a timing library built into the C++ language. The code was inserted before and after the main computational portions of each application in order to accurately read how long each iteration took at varying core quantities.

```
#include <sys/time.h>

struct timeval tim;

gettimeofday(&tim, NULL);
double start = tim.tv_sec + (tim.tv_usec/1000000.0);

gettimeofday(&tim, NULL);
double end = tim.tv_sec + (tim.tv_usec/1000000.0);

printf("Time for core computing: %.4f s\n",end-start);
```

Scaling

- Increasing the number of cores generally decreases execution time
- 32 and 40 cores in OpenMP implementation slows down
- Slowdown at higher core numbers is caused by the use of the fork and join parallelization model by OpenMP
- OpenMP's overhead increases as core numbers increase.
- A fork and join model is illustrated below.



OpenMP vs OpenACC

- OpenMP uses the fork and join method
- OpenACC utilizes several different parallelization protocols to maximize portability
- OpenACC codes experience greater computational overhead and reduced control as opposed to their OpenMP and CUDA counterparts.

References

[1]https://github.com/pathscale/rodinia
[2]http://www.cs.virginia.edu/~skadron/wiki/rodinia/index.php/Needleman-Wunsch
[3]http://www.cs.virginia.edu/~skadron/wiki/rodinia/index.php/Particle_Filter
[4]http://www.openacc.org/
[5]http://openmp.org/wp/

OpenMP

OpenACC
Directives for Accelerators