Is the Intel Xeon Phi right for me?

Fei Mao
SHARCNET
E-mail: feimao@sharcnet.ca
List of Topics:

• What is Xeon Phi?
• Hardware matters!
• Recompile and run?
• Simple codes and benchmarks
• Phi? CPU? GPU?
What is Xeon Phi?

- MIC (Many Integrated Core) architecture
- ~60 (P5110) small X86 cores
- 4x the core count threads (240 threads)
- 8GB GDDR5 memory
- 2 P5110 on Goblin 49
Hardware matters!

- 3 levels of parallelism
  - 1-Thread, 2-SIMD, 3-Cache/Memory
Recompile and run?

- **Execution Modes**
  - Coprocessor \(<----\text{PCI-E BUS}---->\) Host
  - Native mode
  - Offload mode
  - Symmetric mode
Recompile and run?

• Native mode:
  • Recompile your previous code and run!
  • Only adding “–mmic” flag
  • Execution can only be running on Phi
  • SSH to Phi’s OS from host and run
  • Serial job(e.g. disk IO) is painfully slow
Recompile and run?

• **Offload mode:**
  • Add offload directives:
    • `#pragma offload target(mic:0) in(a:length(a))`
      ```
      {  
        massive_parallel_code_running_on_mic()  
      }
      ```
  • **Symmetric mode:**
    • Different executable binary files for host and Phi
    • MPI: 12 ranks on host, 240 ranks natively on Phi
Simple codes

- 2D convolution:
  - OpenMP
  - OpenCL
- MPI:
  - Monte Carlo Estimation of PI
• OpenMP(1 thread for 1 row)

```c
#pragma omp parallel for
for (yOut = halffilter; yOut < imgh; yOut++)
{
    #pragma simd
    for (xOut = halffilter; xOut < imgw; xOut++)
    {
        float sum = 0;
        #pragma unroll
        for (r = -3; r < 4; r++)
        {
            for (c = -3; c < 4; c++)
            {
                sum += inputImage[(yOut+r)*imageWidth+xOut+c] * filter[(r+halffilter)*filterWidth+c+halffilter];
            }
        }
        outputImage[yOut * imageWidth + xOut] = sum;
    }
}
```
• **2D convolution**

• **OpenMP (120M pixels in grey scale, 7x7 filter):**

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Time (ms)</th>
<th>Unrolled Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 x E5-2630(12 cores, AVX)_12threads</td>
<td>504</td>
<td>95</td>
</tr>
<tr>
<td>Phi_60threads</td>
<td>555</td>
<td>165</td>
</tr>
<tr>
<td>Phi_120threads</td>
<td>483</td>
<td>192</td>
</tr>
<tr>
<td>Phi_240threads</td>
<td>579</td>
<td>310</td>
</tr>
</tbody>
</table>

• **CPU 15MB L3 shared, Phi 512KB private L2**
Simple codes

- **OpenCL**
  - 1 work-group maps to 1 HW thread (4 threads each core)
  - Work-items run on SIMD
  - 1D 1 work-group /per row
  - 2D 1 work-item /per pixel
- 1D 1024 workitems: 202ms
- 2D 32x32 workitems: 70ms
- 2D 32x32 with local mem and sync: 82ms
Simple codes

• MPI:
  • 1 rank maps to 1 HW thread
  • Phis and CPUs can communicated
  • Each rank(thread on phi) compute one part

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 x E5-2630(12 cores, AVX) _12threads</td>
<td>6.41</td>
</tr>
<tr>
<td>1 x Phi _120_threads</td>
<td>7.11</td>
</tr>
<tr>
<td>2 x Phi _120_threads</td>
<td>3.74</td>
</tr>
</tbody>
</table>
• **MPI:**

```c
for (point=0; point<num_local_points; point++)
{
    temp = (rand() % (rand_MAX+1));
    p_x = temp / rand_MAX;
    p_x = p_x / num_procs;

    temp2 = (float)id / num_procs;  // id belongs to 0, num_procs-1
    p_x += temp2;

    temp = (rand() % (rand_MAX+1));
    p_y = temp / rand_MAX;

    temp = (rand() % (rand_MAX+1));
    p_z = temp / rand_MAX;

    // Compute the number of points residing inside of the 1/8 of the sphere
    result = p_x * p_x + p_y * p_y + p_z * p_z;

    if (result <= 1)
    {
        inside++;
    }
}
```

• **SIMD:**

```c
#pragma simd private(temp,temp2,pi,p_x,p_y,p_z,result)
for (unsigned int point=0; point<num_local_points; point++)
{
    temp = (rand() % (rand_MAX+1));
    p_x = temp / rand_MAX;
    p_x = p_x / num_procs;

    temp2 = (float)id / num_procs;  // id belongs to 0, num_procs-1
    p_x += temp2;

    temp = (rand() % (rand_MAX+1));
    p_y = temp / rand_MAX;

    temp = (rand() % (rand_MAX+1));
    p_z = temp / rand_MAX;

    // Compute the number of points residing inside of the 1/8 of the sphere
    result = p_x * p_x + p_y * p_y + p_z * p_z;

    int t = 0;
    if (result <= 1)
    {
        t = 1;
    }
    tt[point] = t;
}
#pragma simd reduction(+:inside)
for (int j=0;j<num_local_points;j++)
{
    inside += tt[j];
}
```
Simple codes

- SIMD speed up:

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Time (s)</th>
<th>SIMD (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 x E5-2630(12 cores, AVX)_12threads</td>
<td>6.41</td>
<td>6.7</td>
</tr>
<tr>
<td>1 x Phi_120_threads</td>
<td>7.11</td>
<td>4.24</td>
</tr>
<tr>
<td>2 x Phi_120_threads</td>
<td>3.74</td>
<td>2.14</td>
</tr>
</tbody>
</table>

- Memory bandwidth:
  - PHI : ~160GB/s
  - CPU: ~50GB/s
• CPU v.s. Phi:
  • Code natively run or only adding offload directives
  • Library MKL for large size?
  • 100 more threads?
  • 100 threads and Large vector?
  • 100 threads and Memory bandwidth?
• GPU v.s. Phi
  • Previous CPU code?
  • Learning CUDA/OpenCL?
  • Memory/cache? (more time for optimization)

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phi 32x32</td>
<td>70</td>
</tr>
<tr>
<td>K20 32x32</td>
<td>101</td>
</tr>
<tr>
<td>K20 32x32 local-mem</td>
<td>38</td>
</tr>
</tbody>
</table>
Conclusion

• What is Xeon Phi?
  • Many x86 cores
• Hardware matters!
  • Thread, SIMD, Cache/Mem
• Recompile and run?
  • Native, Offload, Symmetric
• Simple codes and benchmarks
• Phi? CPU? GPU?
References:

- Goblin wiki:
  - https://www.sharcnet.ca/help/index.php/Goblin#The_Phi_Co-processors
- Programming Xeon Phi wiki:
  - https://www.sharcnet.ca/help/index.php/Programming_Xeon_Phi
- Porting CUDA to OpenCL:
  - https://www.sharcnet.ca/help/index.php/Porting_CUDA_to_OpenCL