OpenMP & Parallware

Manuel Arenaz
Appentra Solutions
University of Coruña (Spain)
OUTLINE

➢ Live demo
➢ Why developing Parallware for OpenMP?
➢ Experiments on performance-portability
➢ Conclusions & Future work
OUTLINE

➢ Live demo
➢ Why developing Parallware for OpenMP?
➢ Experiments on performance-portability
➢ Conclusions & Future work
WHY PARALLWARE FOR OpenMP?

- Software modernization through parallelization with MPI+X
  - High-level programming: $X$ is OpenACC or OpenMP

- Parallware is a new tool to assist in parallelization
  - New & disruptive technology for extraction of parallelism
  - Supports OpenMP 2.5 => Interest in extension for accelerators
WHY PARALLWARE FOR OpenMP?

The HPC workflow
WHY PARALLWARE FOR OpenMP?

CLASSICAL DEPENDENCE ANALYSIS  VS.  HIERARCHICAL CLASSIFICATION FOR DEPENDENCE ANALYSIS

intel  PGI  GCC  PARALLWARE

Automatic parallelization  CRAY
WHY PARALLWARE FOR OpenMP?

for(int i=1; i<n; i++) {  
}  

Iteration at source: $I_0 + 1$
Iteration at sink: $I_0 + \Delta I$
Forming an equality gets us: $I_0 + 1 = I_0 + \Delta I$
Solving this gives us: $\Delta I = 1$

for(int i=0; i<n; i++) {  
    for(int j=0; j<n; j++) {  
        for(int k=0; k<n; k++) {  
            A[i+1][j][k] = A[i][j][k+1] + 1;  
        }  
    }  
}  

Forms equalities in each array dimension:  
$I_0 + 1 = I_0 + \Delta I$
$J_0 = J_0 + \Delta J$
$K_0 = K_0 + 1 + \Delta K$
Solutions:  
$\Delta I = 1$  \( \Delta J = 0 \)  \( \Delta K = -1 \)

⇒ Solve systems of mathematical equations to proof the existence of dependences between loop iterations
WHY PARALLWARE FOR OpenMP?

```c
void atmux(double* restrict y, ... , int n) {
    for(int t = 0; t < n; t++)
        y[t] = 0;

    for(int i = 0; i < n; i++) {
        for (int k = row_ptr[i]; k < row_ptr[i+1]; k++) {
            y[col_ind[k]] += x[i] * val[k];
        }
    }
}
```

```
$icc atmux.c -std=c99 -c -O3 -xAVX -Wall -vec-report3 -opt-report3 -restrict -parallel -openmp -guide
icc (ICC) 13.1.1 20130313
...
HPO THREADIZER REPORT (atmux) LOG OPENED ON Fri Sep 25 18:04:15 2015
HPO Threadizer Report (atmux)
atmux.c(9:2-9:2):PAR:atmux: loop was not parallelized: existence of parallel dependence
atmux.c(10:3-10:3):PAR:atmux: potential ANTI dependence on y.
potential FLOW dependence on y.
atmux.c(9:2-9:2):PAR:atmux: LOOP WAS AUTO-PARALLELIZED
atmux.c(12:2-12:2):PAR:atmux: loop was not parallelized: existence of parallel dependence
atmux.c(13:3-13:3):PAR:atmux: loop was not parallelized: existence of parallel dependence
...
WHY PARALLWARE FOR OpenMP?

THERE ARE WELL-KNOWN PARALLELIZATION STRATEGIES THAT APPLY TO “CLASSES OF CODES”
**“SCALAR REDUCTION” CLASS**

```c
sum = 0.0;
for (i = 0; i < N; i++) {
    double x = (i + 0.5) / N;
    sum += sqrt(1 - x * x);
}
pi = 4.0 / N * sum;
```

**“SPARSE REDUCTION” CLASS**

```c
for(t = 0; t < n; t++) {
    y[t] = 0;
}
for(i = 0; i < n; i++) {
    for (k = row_ptr[i]; k < row_ptr[i+1]; k++) {
        y[col_ind[k]] += x[i] * val[k];
    }
}
```

Computation of PI

Product sparse-matrix by vector (ATMUX)
for (i=0; i<n; i++) {
    A[i] = 2000;
}

for (i=1; i<n; i++) {
    B[A[i]] += 2000;
}

r = 0;
for (i=0; i<n; i++) {
    r = r + A[i];
}

r = 0;
for (i=0; i<n; i++) {
    if (A[i] > 0) {
        r = r + B[i];
    }
}

FOCUS ON INFORMATION RELEVANT FOR THE EXTRACTION OF PARALLELISM
Computation of PI

\[ \text{sum} = 0.0; \]
\[ \text{for } (i = 0; i < N; i++) \{ \]
\[ \quad \text{double } x = (i + 0.5) / N; \]
\[ \quad \text{sum} += \sqrt{1 - x \times x}; \]
\[ \} \]
\[ \pi = 4.0 / N \times \text{sum}; \]

double \( f(\text{int } i, \text{int } N) \)
\{ 
\quad \text{return } ((i + 0.5) / N); 
\}

GREAT CHALLENGE FOR PARALLELIZING COMPILERS

TUNING OF THE SYSTEM TO HANDLE SYNTAX VARIATIONS
WHY PARALLWARE FOR OpenMP?

Parallware technology:
- Hierarchical classification for dependence analysis

Advantages:
- Allows incremental detection of syntactical variants of code classes
- Fast & Extensible

Current state of development?
- Effective for first real codes
OUTLINE

➢ Live demo
➢ Why developing Parallware for OpenMP?
➢ Experiments on performance-portability
➢ Conclusions & Future work
## EXPERIMENTS: Performance-Portability

### LAB CODES

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(\pi)</td>
<td>Computation of PI</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Product matrix-vector</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Product sparse matrix-vector</td>
<td></td>
</tr>
</tbody>
</table>
| \[
\begin{bmatrix}
0 & 1 \\
2 & 3
\end{bmatrix}
\] \(\times\) \[
\begin{bmatrix}
1 & 0 \\
0 & 1
\end{bmatrix}
\] | Matrix multiplication |                |
|                | Coulomb law |                |
|                | Mandelbrot sets |                |
|                | Laplace transform |                |

### REAL CODES

<table>
<thead>
<tr>
<th></th>
<th>NPB</th>
<th>SPECaccel</th>
<th>others</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EP</td>
<td>BT</td>
<td>QUAKE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>clvrleaf</td>
<td>CEM_MOM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CEM_FDTD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CG</td>
<td>ShWaters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSP</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

---

WACCPD: Second Workshop on Accelerator Programming Using Directives
## EXPERIMENTS: Performance-Portability

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Description</th>
<th>SLOC</th>
<th>OpenMP Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>PI</td>
<td>Approximation of the PI number by the integration method</td>
<td>8</td>
<td>3.81</td>
</tr>
<tr>
<td>COULOMB</td>
<td>Computation of Coulomb law</td>
<td>26</td>
<td>5.60</td>
</tr>
<tr>
<td>MATMUL</td>
<td>Matrix-Matrix multiplication from dense linear algebra</td>
<td>10</td>
<td>3.18</td>
</tr>
<tr>
<td>MATVEC</td>
<td>Matrix-Vector multiplication from dense linear algebra</td>
<td>8</td>
<td>3.95</td>
</tr>
<tr>
<td>SAXPY</td>
<td>SAXPY operation from dense linear algebra</td>
<td>4</td>
<td>1.14</td>
</tr>
<tr>
<td>PRIME</td>
<td>Computation of prime numbers</td>
<td>11</td>
<td>7.32</td>
</tr>
<tr>
<td>ATMUX</td>
<td>Sparse matrix-vector multiplication from sparse linear algebra</td>
<td>10</td>
<td>2.12</td>
</tr>
<tr>
<td>MANDELBROT</td>
<td>Computation of Mandelbrot sets</td>
<td>39</td>
<td>4.39</td>
</tr>
<tr>
<td>HEATDIFUSSION</td>
<td>Solver of a heat diffusion problem</td>
<td>24</td>
<td>1.92</td>
</tr>
<tr>
<td>LAPLACE</td>
<td>Laplacian smoothing algorithm from digital signal processing (DSP)</td>
<td>30</td>
<td>3.33</td>
</tr>
<tr>
<td>CEM_MOM</td>
<td>Application: Method of moments from computational electromagnetics (CEM)</td>
<td>2108</td>
<td>4.85</td>
</tr>
<tr>
<td>CEM_FDTD</td>
<td>Application: Finite-Difference Time-Domain from comp. electromag. (CEM)</td>
<td>640</td>
<td>3.58</td>
</tr>
<tr>
<td>NPB_EP</td>
<td>Program EP from NAS Parallel Benchmarks (NPB)</td>
<td>181</td>
<td>6.87</td>
</tr>
</tbody>
</table>
### EXPERIMENTS: Performance-Portability

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Parallel code</th>
<th>Reduction type</th>
<th>Explicit transfers</th>
<th>Nested loops</th>
<th>Code refactorization</th>
<th>Deep copy</th>
<th>Iterative solver</th>
<th>Workload</th>
<th>Arithmetic intensity</th>
</tr>
</thead>
<tbody>
<tr>
<td>PI</td>
<td>Partially</td>
<td>Scalar</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Very high</td>
</tr>
<tr>
<td>SAXPY</td>
<td>Fully</td>
<td>IN/OUT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Very low</td>
</tr>
<tr>
<td>MATVECC</td>
<td>Fully</td>
<td>IN/OUT</td>
<td>Yes</td>
<td>Scalarization</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td>Very low</td>
</tr>
<tr>
<td>MATMUL</td>
<td>Fully</td>
<td>IN/OUT</td>
<td>Yes</td>
<td>Scalarization</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td>Medium</td>
</tr>
<tr>
<td>LAPLACE</td>
<td>Partially</td>
<td>Scalar</td>
<td>IN/OUT</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td>Low</td>
</tr>
<tr>
<td>PRIME</td>
<td>Partially</td>
<td>Scalar</td>
<td>IN</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td>Unbalanced</td>
</tr>
<tr>
<td>ATMUX</td>
<td>Partially</td>
<td>Sparse</td>
<td>IN/OUT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unbalanced</td>
</tr>
</tbody>
</table>

Real benchmarks combine the features these simple benchmarks:

1. **NPB_EP** combines features of ATMUX and PRIME
2. **NPB_BT** combines features of ATMUX and MATMUL (ongoing work)
## EXPERIMENTS: Performance-Portability

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Parallel code</th>
<th>Reduction type</th>
<th>Explicit transfers</th>
<th>Nested loops</th>
<th>Code refactoring</th>
<th>Deep copy</th>
<th>Iterative solver</th>
<th>Workload</th>
<th>Arithmetic intensity</th>
</tr>
</thead>
<tbody>
<tr>
<td>PI</td>
<td>Partially</td>
<td>Scalar</td>
<td>IN/OUT</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Very high</td>
</tr>
<tr>
<td>SAXPY</td>
<td>Fully</td>
<td></td>
<td>IN/OUT</td>
<td>Yes</td>
<td>Scalarization</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Very low</td>
</tr>
<tr>
<td>MATVEC</td>
<td>Fully</td>
<td></td>
<td>IN/OUT</td>
<td>Yes</td>
<td>Scalarization</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Very low</td>
</tr>
<tr>
<td>MATMUL</td>
<td>Fully</td>
<td></td>
<td>IN/OUT</td>
<td>Yes</td>
<td>Scalarization</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Medium</td>
</tr>
<tr>
<td>LAPLACE</td>
<td>Partially</td>
<td>Scalar</td>
<td>IN/OUT</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td>Low</td>
</tr>
<tr>
<td>PRIME</td>
<td>Partially</td>
<td>Scalar</td>
<td>IN</td>
<td></td>
<td></td>
<td>Yes</td>
<td></td>
<td>Unbalanced</td>
<td>High</td>
</tr>
<tr>
<td>ATMUX</td>
<td>Partially</td>
<td>Sparse</td>
<td>IN/OUT</td>
<td></td>
<td></td>
<td>Yes</td>
<td></td>
<td>Unbalanced</td>
<td>Very low</td>
</tr>
</tbody>
</table>

Codes with HIGH arithmetic intensity perform well on CPU & GPU

**CPU performs better for small sizes**

**GPU outperforms CPU for large sizes (w/wo resident data)**
### Codes with LOW arithmetic intensity show limited performance on GPU

#### Limited performance on the GPU

**GPU benefits from source code optimizations (e.g. scalarization)**

---

### EXPERIMENTS: Performance-Portability

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Parallel code</th>
<th>Reduction type</th>
<th>Explicit transfers</th>
<th>Nested loops</th>
<th>Code refactorization</th>
<th>Deep copy</th>
<th>Iterative solver</th>
<th>Workload</th>
<th>Arithmetic intensity</th>
</tr>
</thead>
<tbody>
<tr>
<td>PI</td>
<td>Partially</td>
<td>Scalar</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Very high</td>
</tr>
<tr>
<td>SAXPY</td>
<td>Fully</td>
<td></td>
<td>IN/OUT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Very low</td>
</tr>
<tr>
<td>MATVEC</td>
<td>Fully</td>
<td></td>
<td>IN/OUT</td>
<td>Yes</td>
<td>Scalarization</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Very low</td>
</tr>
<tr>
<td>MATMUL</td>
<td>Fully</td>
<td></td>
<td>IN/OUT</td>
<td>Yes</td>
<td>Scalarization</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Medium</td>
</tr>
<tr>
<td>LAPLACE</td>
<td>Partially</td>
<td>Scalar</td>
<td>IN/OUT</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td>Low</td>
</tr>
<tr>
<td>PRIME</td>
<td>Partially</td>
<td>Scalar</td>
<td>IN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unbalanced High</td>
</tr>
<tr>
<td>ATMUX</td>
<td>Partially</td>
<td>Sparse</td>
<td>IN/OUT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unbalanced Very low</td>
</tr>
</tbody>
</table>

---

WACCPD: Second Workshop on Accelerator Programming Using Directives
OUTLINE

- Live demo
- Why developing Parallware for OpenMP?
- Experiments on performance-portability
- Conclusions & Future work
CONCLUSIONS

➢ Technical roadmap for development of Parallware
  ○ Current support is OpenMP 2.5
  ○ Support pragma-based standards OpenACC & OpenMP 4
  ○ Gains in programmability & productivity are clear

➢ Performance-portability needs to be demonstrated
  ○ CPU & GPU offer good performance with high arithmetic intensity
  ○ GPU offers limited performance with low arithmetic intensity
  ○ GPU benefits from source code optimizations that increase arith intensity
  ○ GPU have potential to execute sparse computations efficiently
FUTURE WORK

➢ Development of prototype of Parallware for accelerators
   ○ OpenACC pragmas “parallel”, “loop”, “data copy/copyin/copyout”
   ○ Focus on common capabilities in OpenACC & OpenMP 4

➢ Development of proof-of-concept for “tasking” paradigm
   ○ OpenMP 3 pragmas “task” & “taskwait”
   ○ OpenMP 4 pragmas “task depend(in/out/inout)”

➢ Well-known benchmark suites: SPECaccel, PolyBench
➢ Well-known compilers: Cray, GCC
OpenMP & Parallware

Manuel Arenaz
Appentra Solutions
University of Coruña (Spain)