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

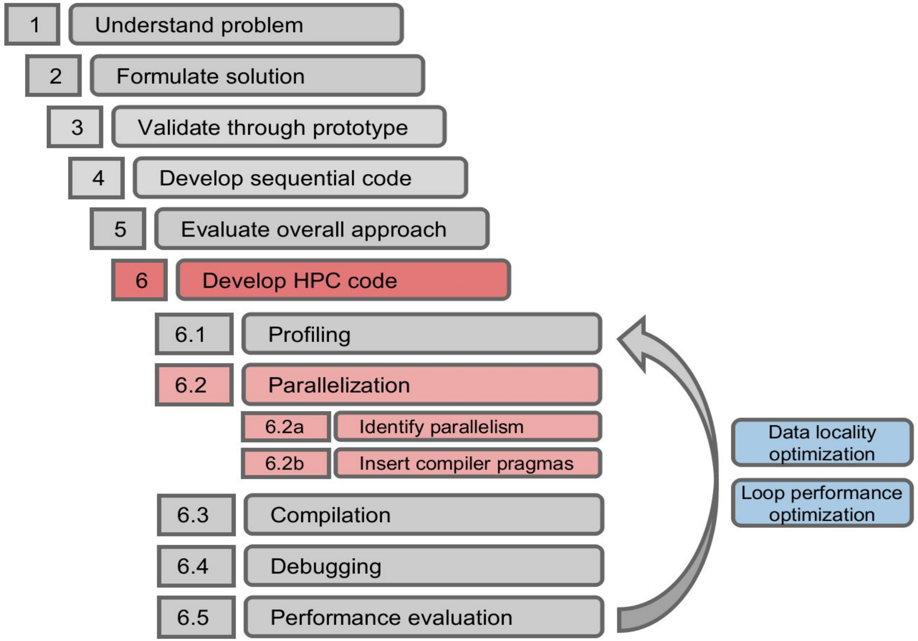
- Live demo
- **Why developing Parallware for OpenMP?**
- Experiments on performance-portability
- Conclusions & Future work

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
CRAY

PARALLWARE
Automatic parallelization

WHY PARALLWARE FOR OpenMP?

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

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:

$I_0 + 1 = I_0 + \Delta I$

$J_0 = J_0 + \Delta J$

$K_0 = K_0 + 1 + \Delta K$

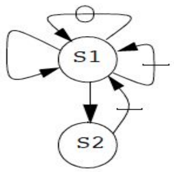
Solutions:

$\Delta I = 1 \quad \Delta J = 0 \quad \Delta K = -1$

⇒ Solve systems of mathematical equations to proof the existence of dependences between loop iterations

WHY PARALLWARE FOR OpenMP?

```
01     void atmux(double* restrict y, … , int n)
08     {
09             for(int t = 0; t < n; t++)
10                     y[t] = 0;
11
12             for(int i = 0; i < n; i++) {
13                     for (int k = row_ptr[i]; k < row_ptr[i+1]; k++) {
14                             y[col_ind[k]] += x[i] * val[k];
15                     }
16             }
17     }
```

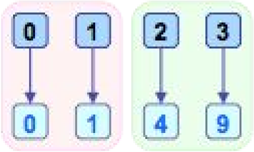


FLOW deps
OUTPUT deps
ANTI deps

```
$ 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
...
```

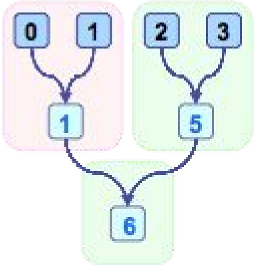
WHY PARALLWARE FOR OpenMP?

parallel for



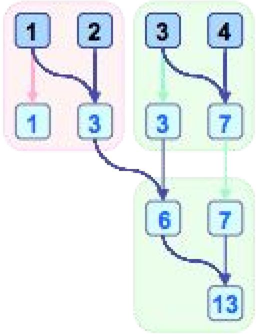
```
for(i=0; i<n; i++) {
    A[i] = i;
}
```

parallel reduction



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

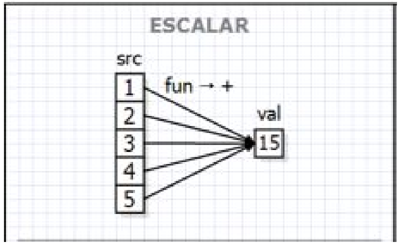
parallel scan



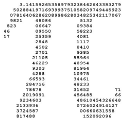
```
for(i=1; i<n; i++) {
    A[i] = A[i] + A[i-1];
}
```

⇒ THERE ARE WELL-KNOWN PARALLELIZATION STRATEGIES THAT APPLY TO “CLASSES OF CODES”

"SCALAR REDUCTION" CLASS

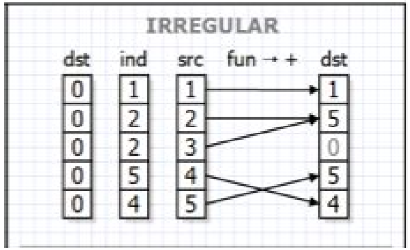



```
sum = 0.0;
for (i = 0; i < N; i++) {
   double x = (i + 0.5) / N;
   sum += sqrt(1 - x * x);
}
pi = 4.0 / N * sum;
```

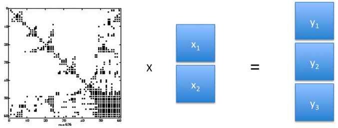


Computation of PI

"SPARSE REDUCTION" CLASS



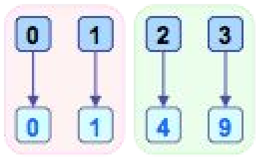

```
for(t = 0; t < n; t++) {
   y[t] = 0;
}
for(i = 0; i < n; i++) {
   for (k = row_ptr[i]; k < row_ptr[i+1]; k++) {
      y[col_ind[k]] += x[i] * val[k];
   }
}
```



Product sparse-matrix by vector (ATMUX)

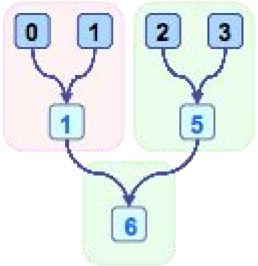
CONTEXTUAL CLASSIFICATION SYSTEM

parallel for




```
for(i=0; i<n; i++) {
    A[i] = 2000;
}
```

parallel reduction

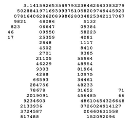



```
for(i=1; i<n; i++) {
    B[A[i]] += 2000;
}
```

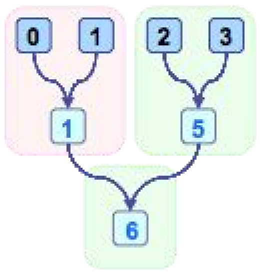
```
r = 0;
for(i=0; i<n; i++) {
    r = r + A[i];
}
```

```
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



TUNING OF THE SYSTEM TO HANDLE SYNTAX VARIATIONS

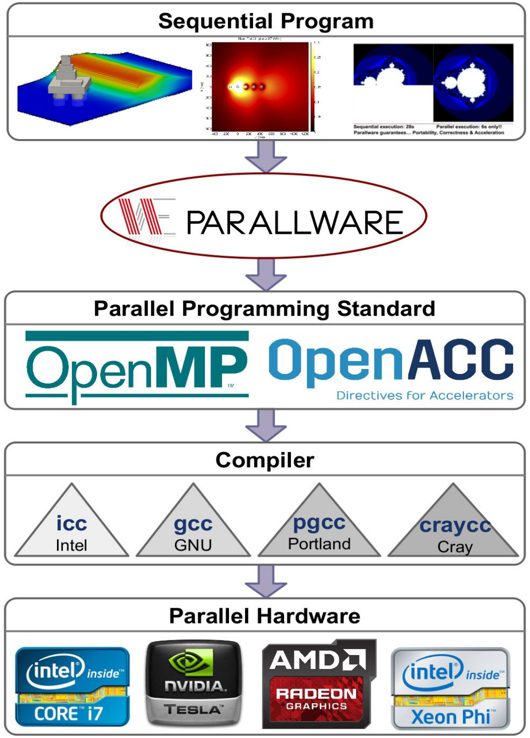
```
sum = 0.0;
for (i = 0; i < N; i++) {
   double x = (i + 0.5) / N;
   sum += sqrt(1 - x * x);
}
pi = 4.0 / N * sum;
```

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

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

GREAT CHALLENGE FOR PARALLELIZING COMPILERS

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

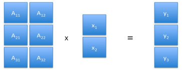
OUTLINE

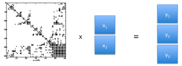
- Live demo
- Why developing Parallware for OpenMP?
- **Experiments on performance-portability**
- Conclusions & Future work

EXPERIMENTS: Performance-Portability

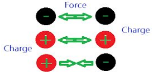
LAB CODES


Computation of PI

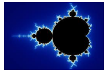

Product matrix-vector


Product sparse matrix-vector


Matrix multiplication



Coulomb law


Mandelbrot sets

Laplace transform

...

REAL CODES

NPB SPECaccel	others
EP	**QUAKE**
BT	**CEM_MOM**
clvrleaf	**CEM_FDTD**
CG	**ShWaters**
CSP	...
...	

EXPERIMENTS: Performance-Portability

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

EXPERIMENTS: Performance-Portability

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

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

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

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)

EXPERIMENTS: Performance-Portability

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

Codes with LOW arithmetic intensity show limited performance on GPU

Limited performance on the GPU

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

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)

SC15 Austin, TX | hpc transforms.

Appentra automatic software acceleration