Data-Parallel Programming
using SaC
lecture 1

F21DP Distributed and Parallel Technology

Sven-Bodo Scholz
The Multicore Challenge

- performance?
- sustainability?
- affordability?
Typical Scenario

- Algorithm
- OpenCL
- VHDL
- μTC
- MPI/OpenMP
Tomorrow’s Scenario

algorithm

OpenCL
VHDL
μTC
MPI/OpenMP
The High-Portability Vision

Algorithm

compiler

OpenCL
VHDL
μTC
MPI/OpenMP

HERIOT WATT UNIVERSITY
What is Data-Parallelism?

Concurrent operations on a single data structure
Data-Parallelism, More Concretely

Formulate algorithms in *space* rather than *time*!

\[
\text{prod} = \text{prod} \cdot (\text{iota}(10)+1)
\]

\[
\text{prod} = 1;
\text{for} (i=1; i<=10; i++) {
    \text{prod} = \text{prod} \cdot i;
}\]

\[
3628800
\]
Why is Space Better than Time?

\[ \prod (\text{iota}(n)) \]

1
2
6
\[ \ldots \]
3628800

1 4 7
2 20 56
6 120 5040
3628800

1 2 \ldots 10
2 \ldots 90
3628800
Another Example: Fibonacci Numbers

```java
if (n<=1) {
    return n;
} else {
    return fib(n-1) + fib(n-2);
}
```
Another Example: Fibonacci Numbers

if( n<=1)  
    return n;  
} else {
    return fib( n-1) + fib( n-2); 
}
Fibonacci Numbers – now linearised!

```
if (n == 0)
    return fst;
else
    return fib(snd, fst+snd, n-1)
```
Fibonacci Numbers – now data-parallel!

$$\text{matprod( genarray( [n], [[1, 1], [1, 0]]) ) [0,0]}$$

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$
Everything is an Array

Think Arrays!

- Vectors are arrays.
- Matrices are arrays.
- Tensors are arrays.
- ....... are arrays.
Everything is an Array

Think Arrays!

- Vectors are arrays.
- Matrices are arrays.
- Tensors are arrays.
- ........ are arrays.

- Even scalars are arrays.
- Any operation maps arrays to arrays.
- Even iteration spaces are arrays.
Multi-Dimensional Arrays

- i
  1 2 3
  shape vector: [3]
  data vector: [1, 2, 3]

j k
  1 2 3 7 8 9
  shape vector: [2, 2, 3]
  data vector: [1, 2, 3, ..., 11, 12]

j
  4 5 6
  shape vector: [ ]
  data vector: [42]
Index-Free Combinator-Style Computations

L2 norm:

\[
\text{sqrt( sum( square( A)))}
\]

Convolution step:

\[
W1 \times \text{shift(-1, A)} + W2 \times A + W1 \times \text{shift(1, A)}
\]

Convergence test:

\[
\text{all( abs( A-B) < eps)}
\]
Shape-Invariant Programming

\[ \text{l2norm}( [1,2,3,4] ) \]

\[ \text{sqrt}( \text{sum}( \text{sqr}( [1,2,3,4])))) \]

\[ \text{sqrt}( \text{sum}( [1,4,9,16]))) \]

\[ \text{sqrt}( 30) \]

\[ 5.4772 \]
Shape-Invariant Programming

\[
\text{l2norm}\left( \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \right) \\
\sqrt{\text{sum}\left( \text{sqr}\left( \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \right) \right)} \\
\sqrt{\text{sum}\left( \begin{bmatrix} 1 & 4 \\ 9 & 16 \end{bmatrix} \right)} \\
\sqrt{\begin{bmatrix} 5 & 25 \end{bmatrix}} \\
\begin{bmatrix} 2.2361 & 5 \end{bmatrix}
\]
Computation of $\pi$

\[ F(x) = \frac{4.0}{1 + x^2} \]

\[ \int_{0}^{1} \frac{4.0}{1 + x^2} \]
Computation of $\pi$

double f(double x)
{
    return 4.0 / (1.0 + x*x);
}

int main()
{
    num_steps = 10000;
    step_size = 1.0 / tod(num_steps);
    x = (0.5 + tod(iota(num_steps))) * step_size;
    y = { iv-> f(x[iv])};
    pi = sum(step_size * y);

    printf("...and $\pi$ is: \%f\n", pi);
    return(0);
}
Programming in a Data-Parallel Style - Consequences

• much less error-prone indexing!
• combinator style
• increased reuse
• better maintenance
• easier to optimise
• huge exposure of concurrency!