Parallelization for Multi-Core

Florian Huonder (fhuonder@hsr.ch)

Department of Computer Science
University of Applied Sciences Rapperswil
Seminar: Program Analysis and Transformation

9. June 2009
Physical Limits

The reason for Multi-Core is called *thermal noise*. Chip vendors are forced to produce
- Faster processing units (size persists)
- Smaller processing units (clock rate persists)

The chip vendors chose *smaller processing units* because *faster processing units* had caused a lot more power consumption.
Challenges

The challenges of concurrent programming

- Concurrent thinking
- Programming overhead
- Determinism
Developers mostly think sequential.

**Quick Sort**

```plaintext
def quicksort(left, right)
    if left < right then
        part := parting(left, right)
        quicksort(left, part - 1)
        quicksort(part + 1, right)
    end
end
```

Concurrent thinking
Learn concurrent thinking

Developers should learn that concurrent programming is the *normal case* and that sequential programming is the *speciality.*
Today concurrent programming has too much overhead. Developers have to deal with stuff like

- Synchronization
- Scheduling
Determinism

- It is hardly possible to prove determinism
- Bugs only appear in certain compositions (how to test?)
- How to debug?

Algorithm design

Proper algorithm design is a key point in reaching a certain extent of determinism.
The way concurrency is realised today.
- Parallel Random Access Machine (PRAM)
- Message Passing Interface (MPI)
- OpenMP
- Java Language Features
The Parallel Random Access Machine is a model that aids in algorithm design.

- Pseudo code, *pardo* keyword
- Shared memory model
- A set of $n$ processors
Quick Sort

```java
func quicksort(left, right)
    if left < right then
        part := parting(left, right)
        quicksort(left, part - 1)
        quicksort(part + 1, right)
    end
end
end
```
The solution worked out with PRAM can then be transformed into a target language, e.g. Java.
MPI is a de facto standard for concurrent programming in C / C++ and Fortran.

- Data types
- Point to Point communication
- External interfaces
- …
Simple idea

Process A sends a message to process B

On this way all processes can communicate with each other and balance the workload of a certain problem.
OpenMP follows in contrast to MPI another model, the PRAM.

Also for C / C++ and Fortran

Declarative programming style
Example

```c
int main(int argc, char **argv) {
    const int N = 100000;
    int i, a[N];

    #pragma omp parallel for
    for (i = 0; i < N; i++){
        a[i] = 2 * i;
    }

    return 0;
}
```
Java supports, as MPI, a very static programming model for concurrency.

- Threads
- Synchronized
- Wait/Notify
- Volatile
New Approaches

High Performance Computing

How to fulfill the performance requirements of tomorrow’s applications?

- Chapel (Cray Inc.)
- Fortress (Sun Microsystems)
- X10 (IBM)
New Approaches

All these new HPC languages follow the PRAM model. They introduce a new level of abstraction.

Abstraction level

- X10, Fortress, Chapel
- Threads, Synchronization, Locks
- Runtime (e.g. JVM)
- Operating System
Chapel

- Chapel is developed by the Cray Inc.
- Syntax based on Java, C#, C, C++, Cray MTA, Modula and Ada
Concurrency features are:
- Spawning single and multiple activities
- Synchronizing activities
- Concurrent for-loop
- atomic blocks
Chapel

Spawning a single activity

```chapel
begin
  something1();
  something2();
  something3();
end
```

Spawning multiple activities

```chapel
cobegin
  something1();
  something2();
  something3();
end
```
Chapel

Synchronizing activities

```chapel
sync {
    begin a = something1();
    b = something2();
}

c = a + b;
```
Concurrent for-loop

```plaintext
coforall x in col {
    doSomething(x);
}
```
Chapel

Atomic block

```plaintext
atomic{
    something1();
    something2();
    something3();
}
```
Fortress

- Fortress is developed by Sun Microsystems
- Complete new syntax (mathematical notation)
- Aimed at scientists and engineers
Concurrent features are
- Concurrent for-loops
- Tuples
- Blocks
- All statements
Concurrent for-loop

```
for i ← 1:10 do
  print(i " ");
end
```

Example output: 7 5 3 9 1 2 4 8 6
To force a for-loop to run sequential, a special generator has to be used

**Sequential for-loop**

```plaintext
for i ← sequential(1:10) do
    print(i " ");
end
```

Example output: 1 2 3 4 5 6 7 8 9
Fortress

**Tuples**

\[(f, s, t) = (\text{first}(x), \text{second}(x), \text{third}(x))\]
Block

```
do
  f = first(x)
do also
  s = second(x)
do also
  t = third(x)
end
```
Fortress

Blocks

Fortress tries by default to execute all statements concurrently. To force Fortress to run particular statements sequential, they have to be put into a `do - end` block.
Concurrency features are

- Spawning a single activity (statements and code-blocks)
- Synchronizing activities
- atomic blocks
spawning a statement

```java
async for (int i = 0; i < 3; i++) {
    System.out.println("A:" + i + ", ");
}
for (int i = 0; i < 3; i++) {
    System.out.println("B:" + i + ", ");
}
```

Example output: A:0, A:1, B:0, A:2, B:1, B:2,
spawning a code-block

```java
future<int> f = future {something1();}
int r2 = something2();
int r1 = f.force();
result = r1 + r2;
```
Synchronizing activities

```java
finish {
    async for (int i = 0; i < 3; i++) {
        a += i;
    }
    for (int i = 0; i < 3; i++) {
        b += i;
    }
}
c = a + b;
```
Atomic blocks

```plaintext
atomic{
    something1();
    something2();
}
```
Demo …
In Comparison

### Area of application
- Chapel: ?
- Fortress: Scientific
- X10: Java developers

### Syntax
- Chapel: C, C++, ...
- Fortress: Completely new
- X10: Java
In Comparison

- Fortress will find its place in a scientific niche
- X10 and Chapel both aim at Java/C++/C# community

**Winner**

In my opinion the winners will be Fortress, in its scientific niche, and X10 because it is based on Java that enables a faster spread.

Florian Huonder (fhuonder@hsr.ch)
Today’s approach with *Threads, Synchronized Blocks* and so on is not en vogue among developers → it is hard, cumbersome and error-prone.

New languages have to come onto the stage to master the future, Chapel, X10 and Fortress sound very promising.