A parallel programming language extension for Java

Submit and upvote questions: sli.do #geecon2022

Patrick Viry - `neograms.com`

geecon
PRAGUE
24-25 Oct 2022

Extending general purpose languages (GPLs)

Domain-specific languages (DSLs) tend to grow into General Purpose Languages (GPLs)

Ateji : a (defunct) software startup to provide DSLs as extensions to GPLs

A dozen experiments, two products :

- OptimJ (logical variables, constraints, objectives, …)
- Ateji PX (parallel programming) ← this talk

ATEJI®

Language tools for software productivity

Models of parallelism (1/2)

Threads

- a hardware-level concept, not a high-level language construct
- most multi-threaded programs are buggy

Edward A. Lee, “The Problem with Threads”, Berkeley Technical Report No. UCB/EECS-2006-1.

Task parallelism

- fork/join (Unix), “spawn” a task, …
- Dedicated languages such as Cilk and Erlang.

Message passing

- MPI (supercomputers, scientific computing)

Models of parallelism (2/2)

OpenMp / OpenACC

- Data parallelism
- Write sequential code in C/C++, then add parallel directives

CCS / Pi-calculus

- Theoretical, “lambda-calculus for parallelism”
- Compositional operators ← *that’s the main idea*

Ateji PX is based on composing parallel branches. This is essential for writing structured and analyzable code, that programmers and tools can reason about. It also makes parallel code more intuitive, closer to the way we think about and understand the notion of parallelism.

It is based on a sound theoretical foundation and can easily emulate all the above models.

Parallelism at the language level

a = a+1 ; b = b+1

first increment a, then increment b

a = a+1 || b = b+1

increment a and increment b, in no particular order

"Compositional"
(ie. not "start a thread" or "spawn a task")

What we can express with the compositional parallel operator

geecon

Patrick Viry

PRAGUE
24-25 Oct 2022

Data parallelism

Add quantifiers to the parallel composition operator (generators and filters)

```
[
     || (int i:N, int j:N, if i+j<N) matrix[i][j]++;
]
```

“Parallel for” to easily parallelize existing sequential code.

```
for(int i : I) {            ||            for||(int i : I) {
    ...                    ===>               ...
}                                         }
```

Recursive parallelism

Each recursive call creates parallel branches.

```
int fib(int n) {
    if(n <= 1) return 1;
    int fib1, fib2;
    // recursively create parallel branches
    [
        || fib1 = fib(n-1);
        || fib2 = fib(n-2);
    ]
    return fib1 + fib2;
}
```

Speculative parallelism

```
int[] sort(int[] array) {
    [
        || return mergeSort(array);
        || return quickSort(array);
    ]
}
```

Difficult to get right when using threads or tasks :

- Terminating branches properly with non-local exits (return, and possibly exceptions)

Parallel reductions

Values

```
int sumOfSquares = `+ for||(int i : N) (i*i);
// sumOfSquares = 0*0 + 1*1 + ... + (N-1)*(N-1)
```

Collections

```
Set<String> s = set() for||(Person p : persons) p.name;
// s = { p1.name, ..., pN.name }
```

This was before Java streams…
Streams are an operational description, this is an algebraic description : `+ and set() are monoids
Ateji PX also has algebraic collections… but that’s for another time.

Message passing

geecon

Patrick Viry

PRAGUE
24-25 Oct 2022

Sending and receiving messages

```
// declare a channel visible by both branches, and instanciate it
Chan<String> chan = new Chan<String>();
[
     // send a value over the channel
     || chan ! "Strč prst skrz krk";
     // receive a value from the channel, and print it
     || chan ? s; System.out.println(s);
]
```

Non-determinism

Read two values from two channels, in no particular order:

```
[ in1 ? Value1; || in2 ? Value2; ]
```

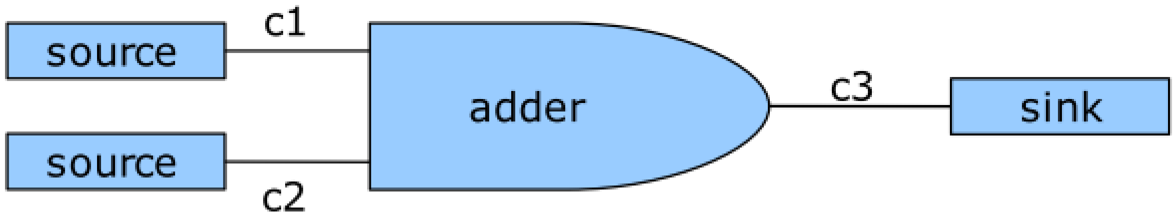
Either read a value from chan1 and print it, or read a value from chan2 and print it.

```
select {
    when chan1 ? v : println("1: " + v);
    when chan2 ? v : println("2: " + v);
}
```

Typical of a server accepting connections.
Similar to the Unix select() system call and the Java NIO Selector API.

Data-Flow, Actors, ...

```
void adder(Chan<Integer> in1, Chan<Integer> in2,
           Chan<Integer> out) {
    for(;;) {
        int value1, value2;
        [ in1 ? value1; || in2 ? value2; ];
        out ! value1 + value2;
}}
```



Star operator

As many parallel branches as needed…

```
[
    ||* channel ? request : process(request);
]
```

Whenever a value is received, a new branch is created on-demand to handle it.

Distributed parallelism

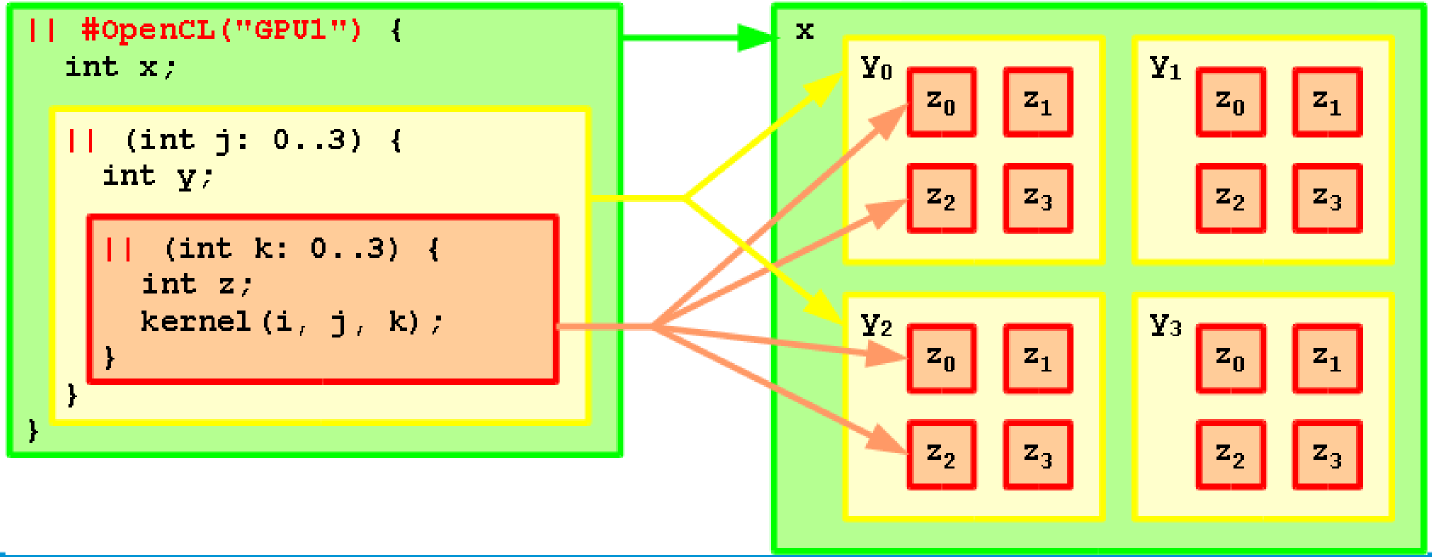
geecon

Patrick Viry

PRAGUE
24-25 Oct 2022

GPU support

- Locate branches with the #OpenCL annotation
- Only a small subset of Java. Same code works on CPU and GPU.
- Explicit message passing with ! and ? operators.
- GPU memory hierarchy as lexical scope.



Distributed computing support

- Locate branches with the #IP annotation
- Explicit message passing with ! and ? operators.

```
// declare a channel visible by both branches, and instanciate it
Chan<String> chan = new Chan<String>();
[
    // send a value over the channel
    || #IP("192.168.0.2") chan ! "Hello";
    // receive a value from the channel, and print it
    || #IP("192.168.0.3") chan ? s; System.out.println(s);
]
```

Makes explicit the (often implicit) structure of distributed systems.

Correctness of parallel programs

Patrick Viry

Correctness of parallel programs

No data races, no deadlocks, etc…

Forget “thread safety”

- Code correctness requires intricate thinking and inspection of the whole program

Compositional operators

- Local: Prove correctness at the point of composition.
- Specific: Prove correctness only for this specific composition
- Can be (somewhat) automated

Based on a theoretical model

- Pi-calculus : all our operators are defined there
- When not sure about the expected behavior, refer to pi-calculus

Performance

Performance

- At least as efficient as threads
- Easier to focus on the algorithm, to debug, to experiment different ways of slicing a problem

```java
final int nThreads = System.getAvailableProcessors();
final int blockSize = I / nThreads;
Thread[] threads = new Thread[nThreads];
for(int n=0; n<nThreads; n++) {
  final int finalN = n;
  threads[n] = new Thread() {
    void run() {
        final int beginIndex = finalN*blockSize;
        final int endIndex = (finalN == (nThreads-1))?
                             I :(finalN+1)*blockSize;
        for( int i=beginIndex; i<endIndex; i++) {
          for(int j=0; j<J; j++) {
            for(int k=0; k<K; k++) {
              C[i][j] += A[i][k] * B[k][j];
    }}}}};
    threads[n].start();
}
for(int n=0; n<nThreads; n++) {
  try {
    threads[n].join();
  } catch (InterruptedException e) {
    System.exit(-1);
  }
}
```

Parallel Matrix Multiplication with Java threads

```java
for||(int i : I) {
    for(int j : J) {
        for(int k : K) {
            C[i][j] += A[i][k] * B[k][j];
        }
    }
}
```

Parallel Matrix Multiplication with Ateji PX

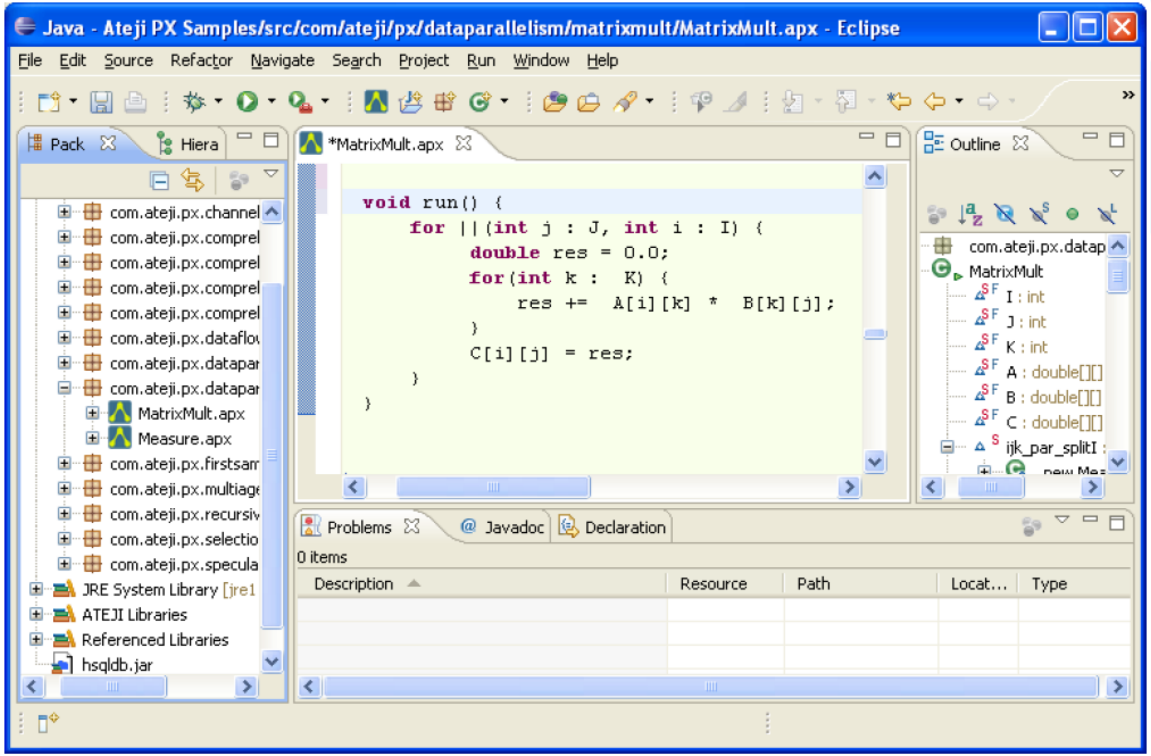
Performance

- Implementation will benefit from project Loom / JEP 425
  (lightweight fibers / continuations vs. heavy threads)
  Erlang-style programming becomes possible

- Remember the example of the adder : parallel programming is not just about performance.

Tooling support

Outdated...



Java - Ateji PX Samples/src/com/ateji/px/dataparallelism/matrixmult/MatrixMult.apx - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Pack
Hiera
com.ateji.px.channel
com.ateji.px.comprel
com.ateji.px.comprel
com.ateji.px.comprel
com.ateji.px.comprel
com.ateji.px.dataflo
com.ateji.px.datapar
com.ateji.px.datapar
MatrixMult.apx
Measure.apx
com.ateji.px.firstsam
com.ateji.px.multiage
com.ateji.px.recursiv
com.ateji.px.selectio
com.ateji.px.specula
JRE System Library [jre1
ATEJI Libraries
Referenced Libraries
hsqldb.jar
*MatrixMult.apx
void run() {
for ||(int j : J, int i : I) {
double res = 0.0;
for(int k : K) {
res += A[i][k] * B[k][j];
}
C[i][j] = res;
}
}
Outline
com.ateji.px.datap
MatrixMult
I : int
J : int
K : int
A : double[][]
B : double[][]
C : double[][]
ijk_par_splitI
Problems
Javadoc
Declaration
0 items
Description
Resource
Path
Locat...
Type

Takeaways / Food for thought

- Remember one word : **“Compositional”**
- Concise and intuitive
- Language extensions are great but difficult :
  - Tooling support
  - Adoption
  - Compatibility between different extensions

Learn more

Lots of code samples on **`github.com/PatrickViry/Ateji`**

- Tooling support out of date

You're welcome to reuse / extend

These ideas apply to most languages, not just Java

- Implementation of the synchronization core available

Contact **`patrick.viry@neograms.com`**