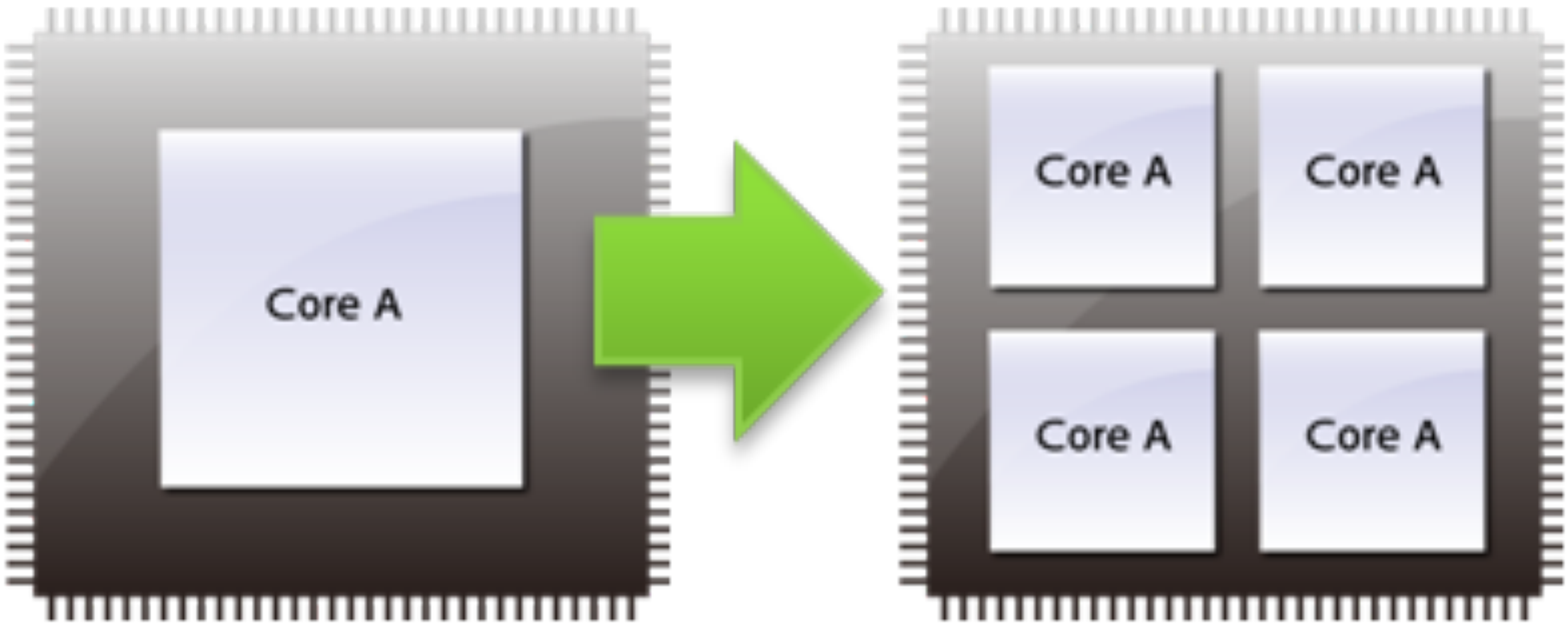
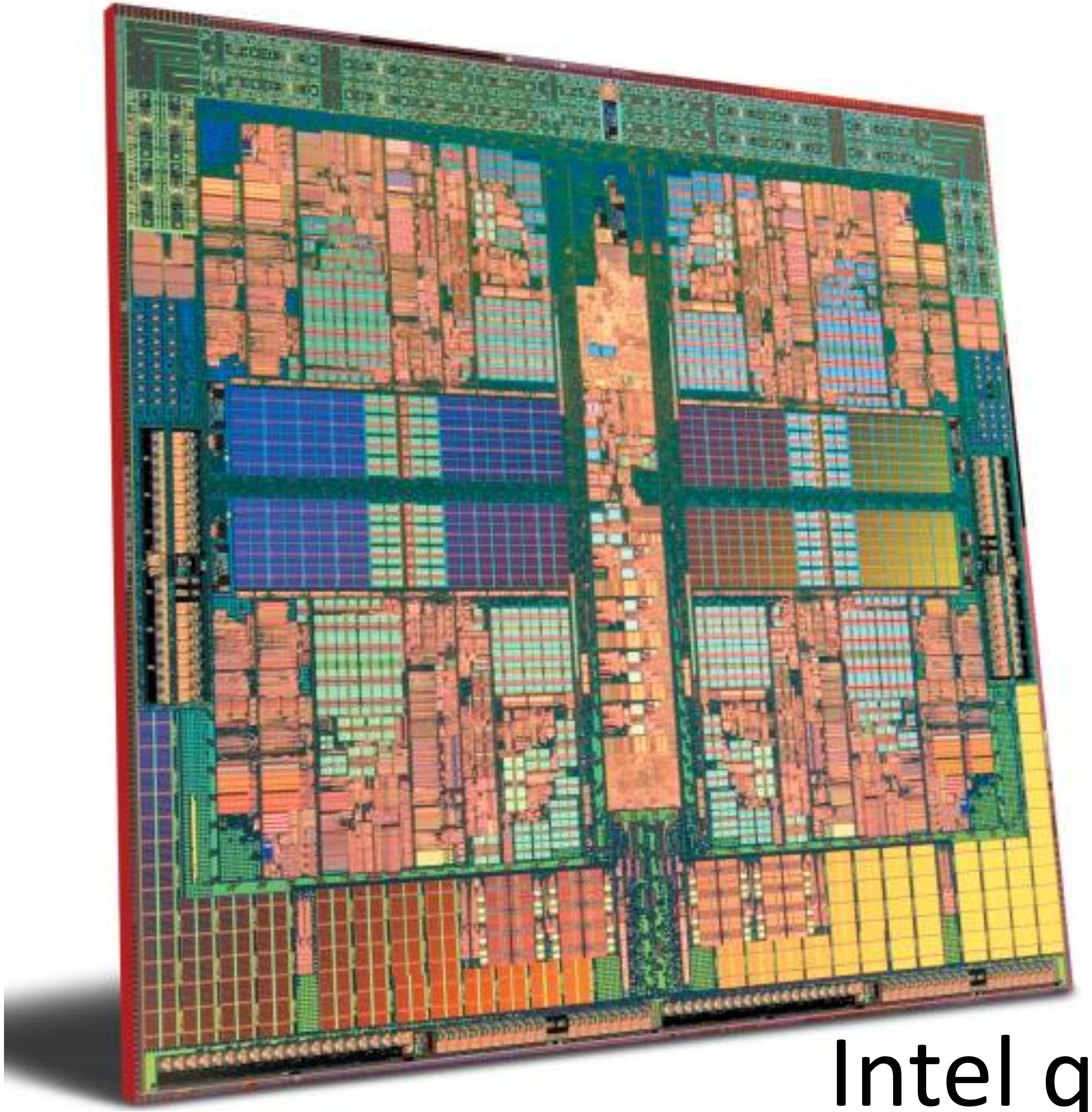
Multi-Core Data Flow Analysis

Jeremy Singer
Glasgow

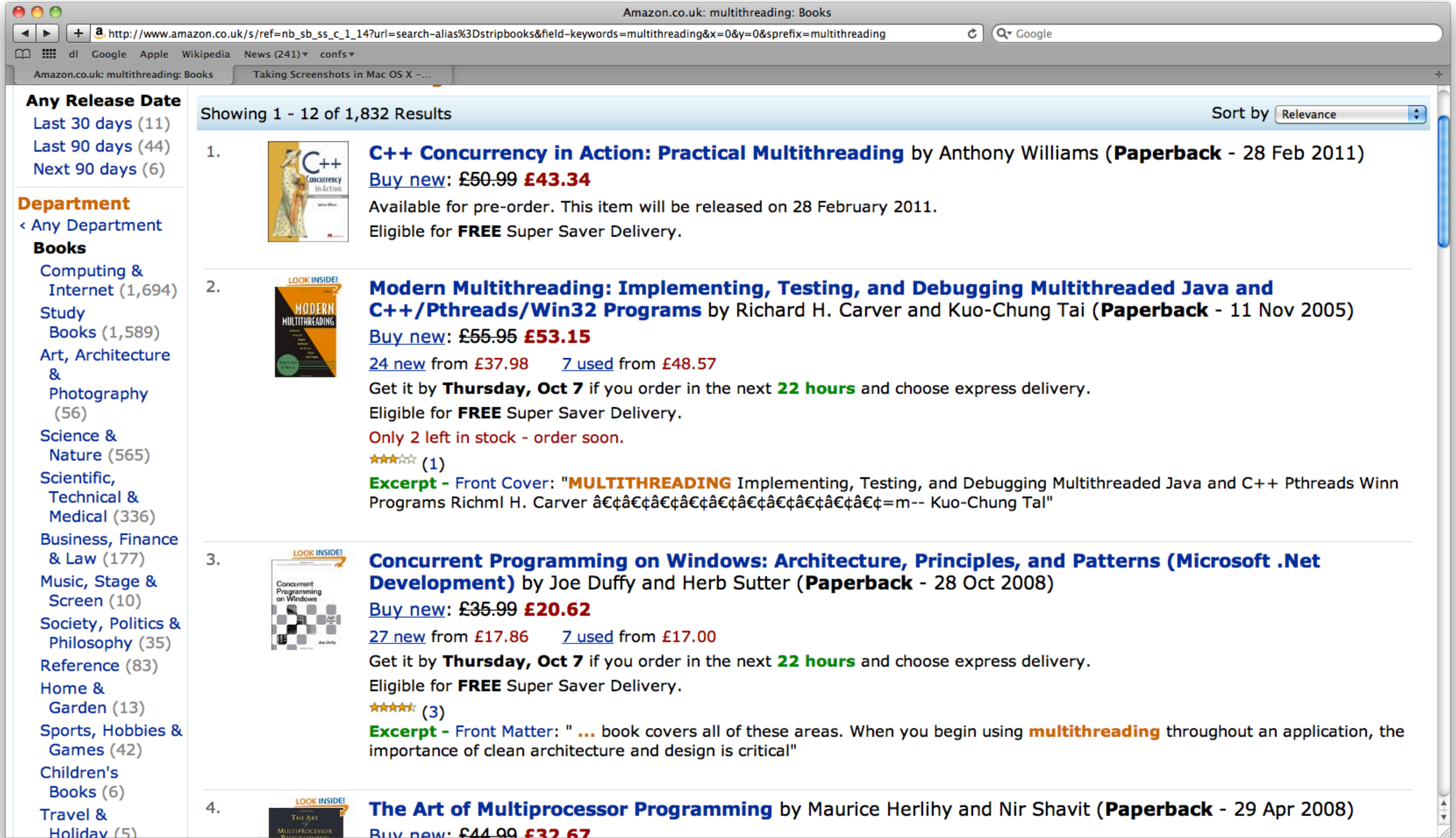
Martin Ward
De Montford


Core A
Core A
Core A
Core A
Core A



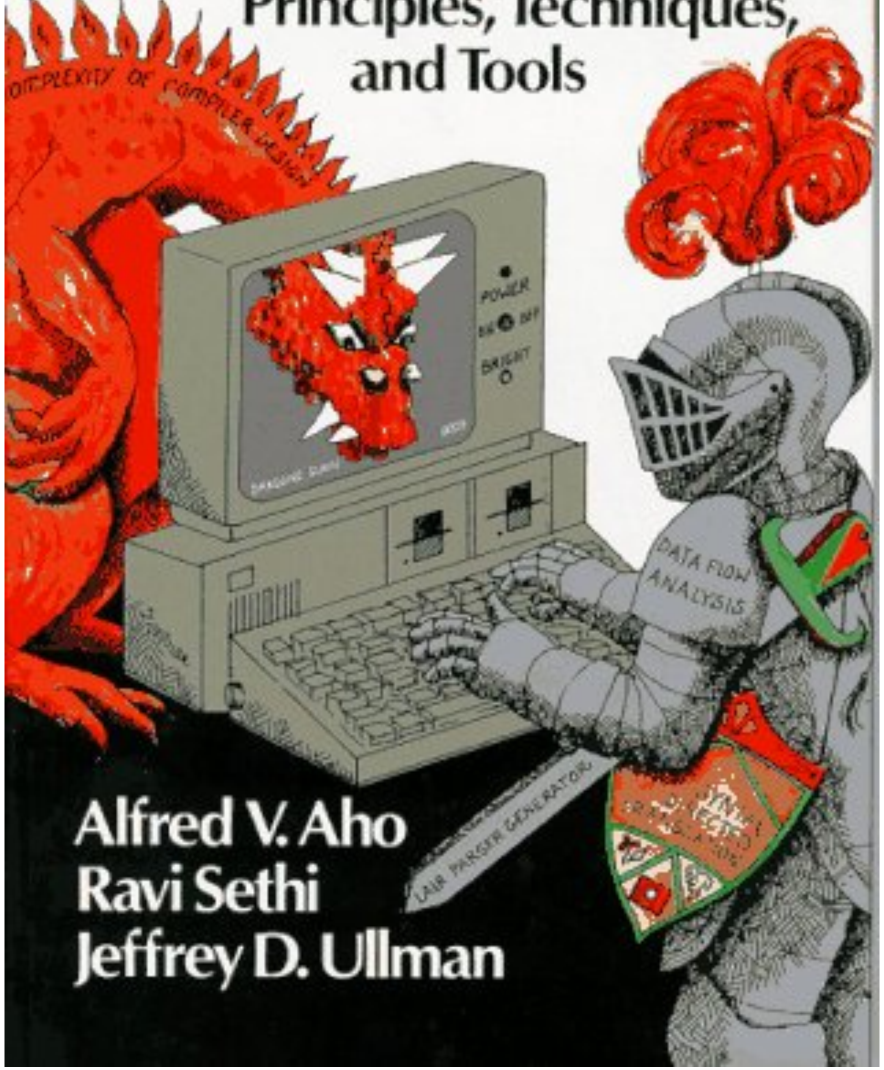
Intel quad-core


NASA
sgi

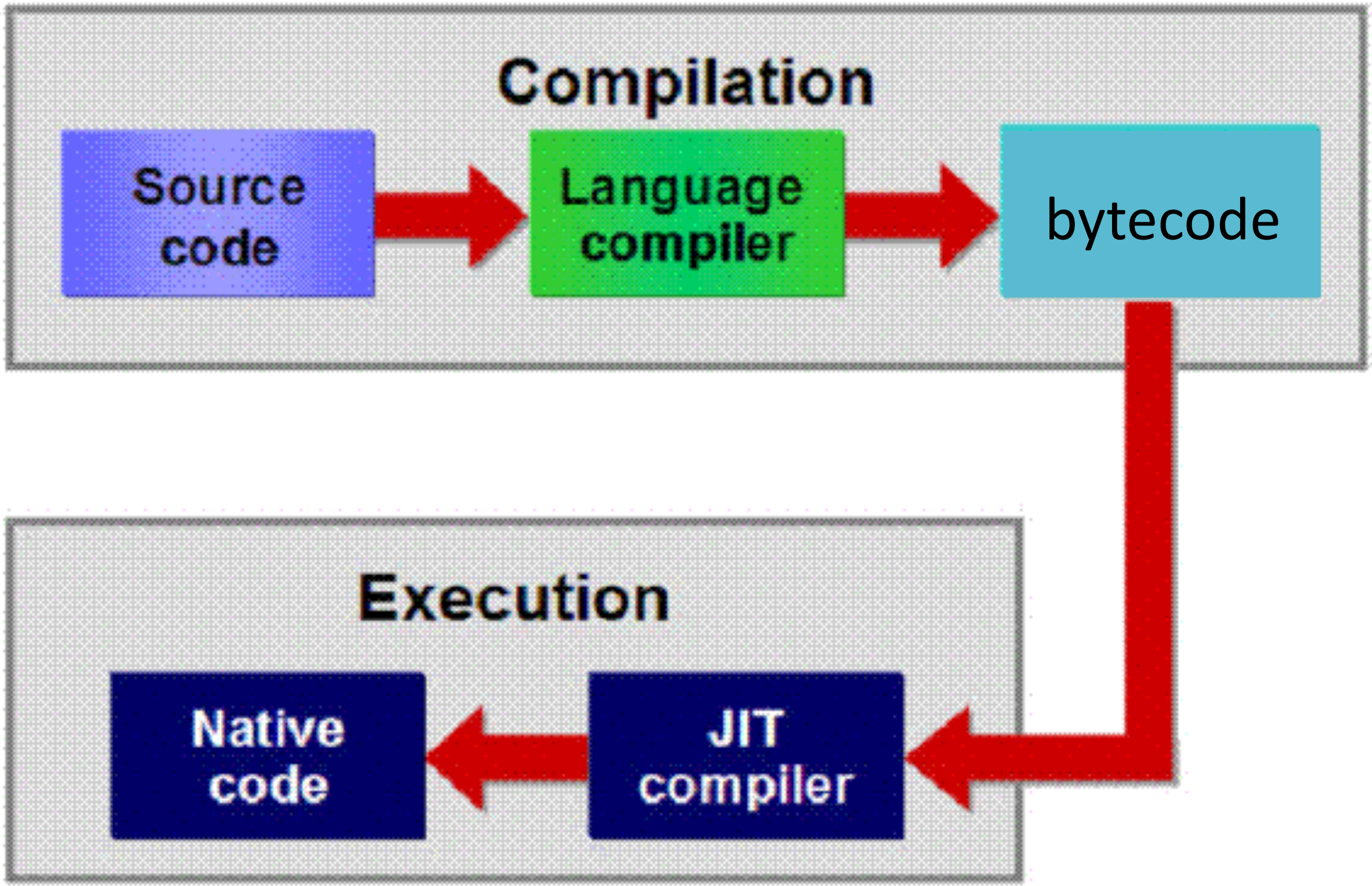

Amazon.co.uk: multithreading: Books
http://www.amazon.co.uk/s/ref=nb_sb_ss_c_1_14?url=search-alias%3Dstripbooks&field-keywords=multithreading&x=0&y=0&sprefix=multithreading
Google
dl Google Apple Wikipedia News (241) confs
Amazon.co.uk: multithreading: Books
Taking Screenshots in Mac OS X -...
Any Release Date
Last 30 days (11)
Last 90 days (44)
Next 90 days (6)
Department
‹ Any Department
Books
Computing & Internet (1,694)
Study Books (1,589)
Art, Architecture & Photography (56)
Science & Nature (565)
Scientific, Technical & Medical (336)
Business, Finance & Law (177)
Music, Stage & Screen (10)
Society, Politics & Philosophy (35)
Reference (83)
Home & Garden (13)
Sports, Hobbies & Games (42)
Children's Books (6)
Travel & Holiday (5)
Showing 1 - 12 of 1,832 Results
Sort by Relevance
1. C++ Concurrency in Action: Practical Multithreading by Anthony Williams (Paperback - 28 Feb 2011)
Buy new: £50.99 £43.34
Available for pre-order. This item will be released on 28 February 2011.
Eligible for FREE Super Saver Delivery.
2. Modern Multithreading: Implementing, Testing, and Debugging Multithreaded Java and C++/Pthreads/Win32 Programs by Richard H. Carver and Kuo-Chung Tai (Paperback - 11 Nov 2005)
Buy new: £55.95 £53.15
24 new from £37.98 7 used from £48.57
Get it by Thursday, Oct 7 if you order in the next 22 hours and choose express delivery.
Eligible for FREE Super Saver Delivery.
Only 2 left in stock - order soon.
(1)
Excerpt - Front Cover: "MULTITHREADING Implementing, Testing, and Debugging Multithreaded Java and C++ Pthreads Winn Programs Richml H. Carver â€¢â€¢â€¢â€¢â€¢â€¢â€¢â€¢â€¢â€¢â€¢=m-- Kuo-Chung Tal"
3. Concurrent Programming on Windows: Architecture, Principles, and Patterns (Microsoft .Net Development) by Joe Duffy and Herb Sutter (Paperback - 28 Oct 2008)
Buy new: £35.99 £20.62
27 new from £17.86 7 used from £17.00
Get it by Thursday, Oct 7 if you order in the next 22 hours and choose express delivery.
Eligible for FREE Super Saver Delivery.
(3)
Excerpt - Front Matter: " ... book covers all of these areas. When you begin using multithreading throughout an application, the importance of clean architecture and design is critical"
4. The Art of Multiprocessor Programming by Maurice Herlihy and Nir Shavit (Paperback - 29 Apr 2008)
Buy new: £44.99 £32.67

Compilers

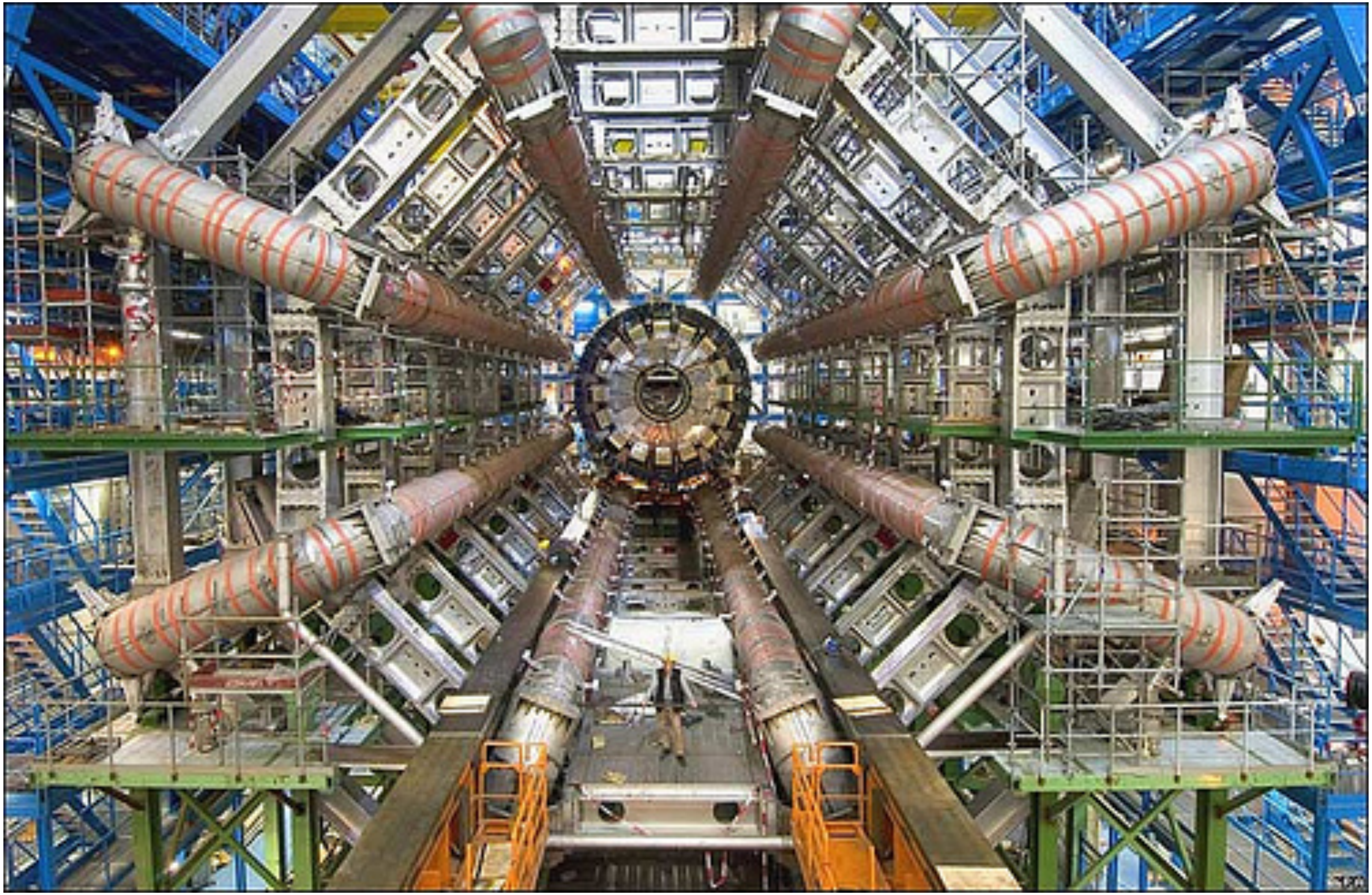
Principles, Techniques, and Tools

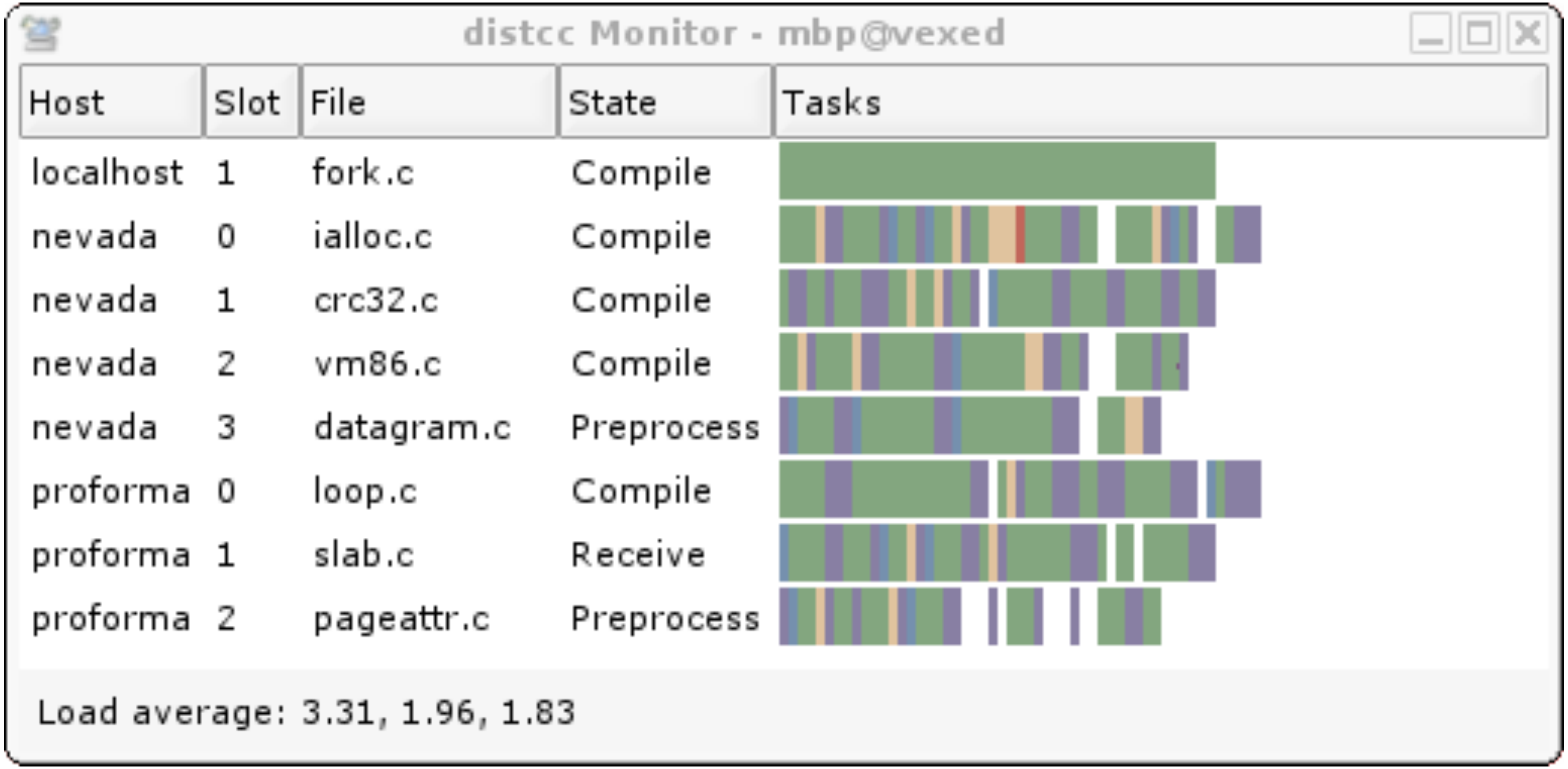


Alfred V. Aho
Ravi Sethi
Jeffrey D. Ullman


Compilation
Source code
Language compiler
bytecode
Execution
Native code
JIT compiler


Java Setup - Welcome
Java™
Sun
Welcome to Java™
Java will make your Internet experience richer. Whether you are playing games or music, getting email on your mobile phone, checking out a webcam, learning about the universe, or anything in between, Java can make it better.
View License Agreement...
You must accept the license agreement by clicking the Accept button to download the product.
Show advanced options panel
Decline
Accept >




distcc Monitor - mbp@vexed
Host
Slot
File
State
Tasks
localhost 1 fork.c Compile
nevada 0 ialloc.c Compile
nevada 1 crc32.c Compile
nevada 2 vm86.c Compile
nevada 3 datagram.c Preprocess
proforma 0 loop.c Compile
proforma 1 slab.c Receive
proforma 2 pageattr.c Preprocess
Load average: 3.31, 1.96, 1.83

GCC

LLVM

Static Single Assignment Form

- A program is defined to be in SSA form if each variable is a target of exactly one assignment statement in the program text.

Static Single Assignment Form

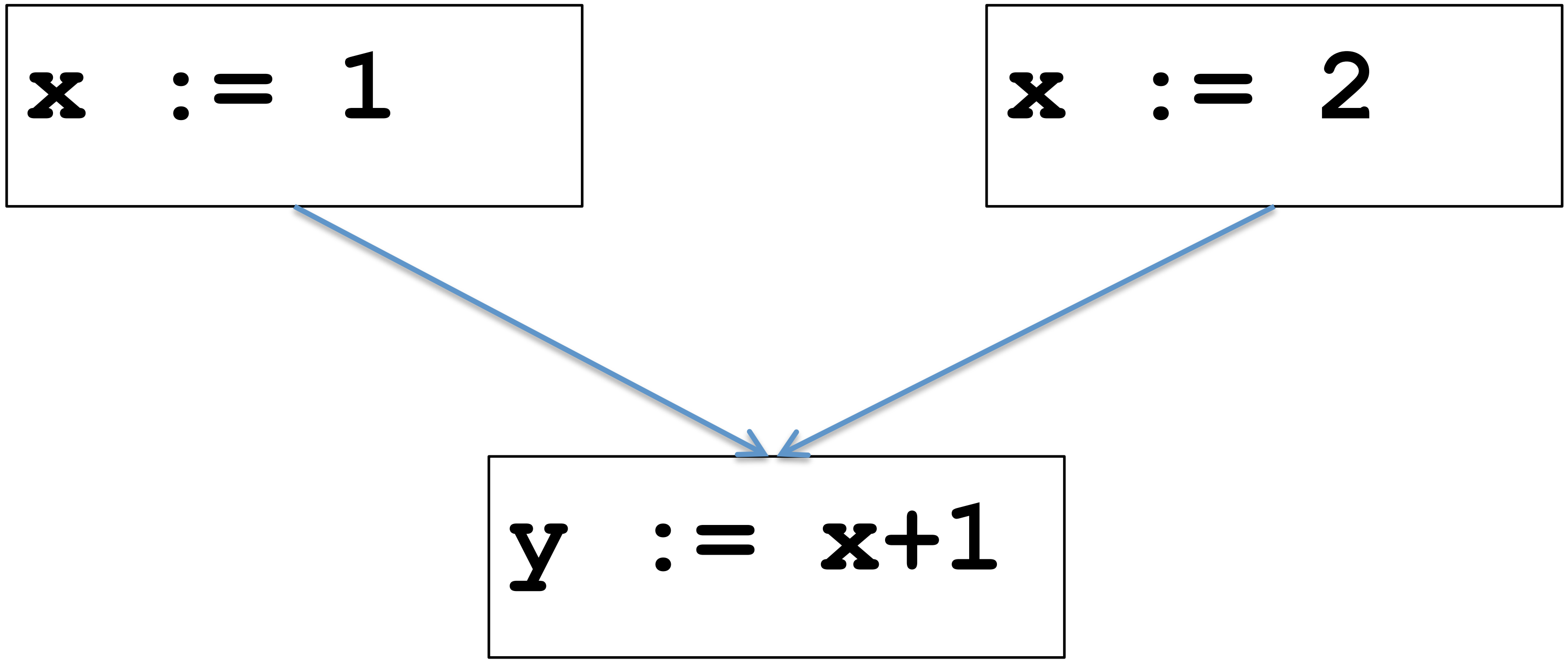
- A program is defined to be in SSA form if each variable is a target of exactly one assignment statement in the program text.

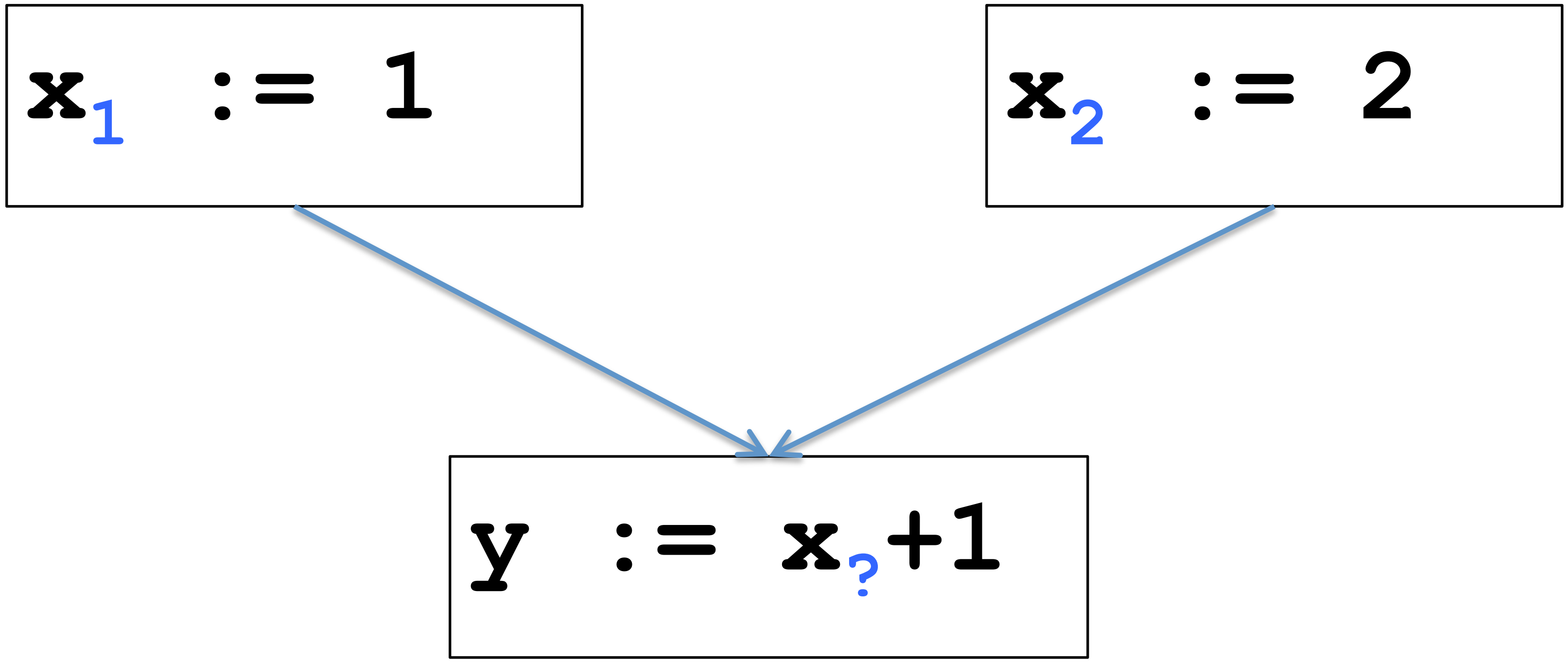
Static Single Assignment Form

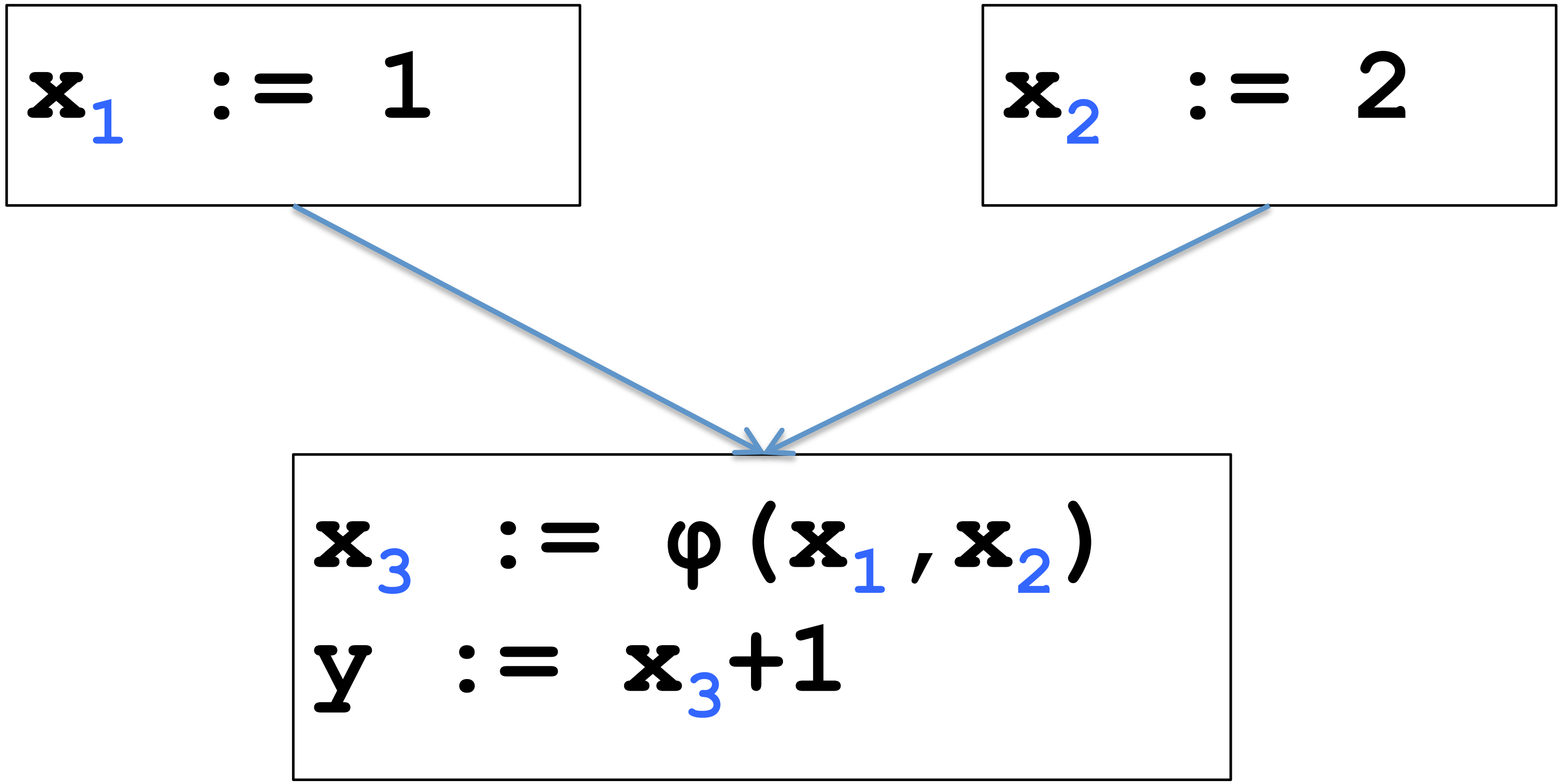
- A program is defined to be in SSA form if each variable is a target of exactly one assignment statement in the program text.

```
x := 1
y := 2
x := y
```

$$x_1 := 1$$
$$y_1 := 2$$
$$x_2 := y_1$$


x := 1
x := 2
y := x+1


x1 := 1
x2 := 2
y := x?+1


x_1 := 1
x_2 := 2
x_3 := φ(x_1,x_2)
y := x_3+1

SSA Construction Algorithm

- start with control flow graph derived from program
- Variable names from orig program, and compiler-generated temporaries
- produce control flow graph with SSA property

Two phases for construction

1. insert **φ**-functions
2. rename variables

High-level φ-function insertion

- for each variable *x*
  - for each definition of *x*
    - follow control flow paths from *def*,
    - at each merge point where *def* is no longer the only definition of *x* in scope, insert a φ-function for *x*

Actual φ-function insertion algorithm

Algorithm 1 Classical ϕ-function insertion algorithm

```
W ← {}
for all v : variable names in original program do
    for all d : definition statements of variable v do
        let B be the basic block containing d
        W ← W ∪ {B}
    end for
    while W ≠ {} do
        remove a basic block X from W
        for all Y : block ∈ DF(X) do
            if Y does not contain a φ-function for v then
                add v ← φ(...) at start of Y
                if Y has not already been processed in W then
                    W ← W ∪ {Y}
                end if
            end if
        end for
    end while
end for
```

Actual φ-function insertion algorithm

Algorithm 1 Classical ϕ-function insertion algorithm

```
W ← {}
for all v : variable names in original program do
    for all d : definition statements of variable v do
        let B be the basic block containing d
        W ← W ∪ {B}
    end for
    while W ≠ {} do
        remove a basic block X from W
        for all Y : block ∈ DF(X) do
            if Y does not contain a φ-function for v then
                add v ← φ(...) at start of Y
                if Y has not already been processed in W then
                    W ← W ∪ {Y}
                end if
            end if
        end for
    end while
end for
```

parallel

DOALL

privatize W

synchronize updates to CFG

ENDDO

High-level renaming algorithm

- have an int counter for each orig var: Count(v)
- have a stack of new vars for each orig var: Stack (v)
- go through program statements in order
- At def of x, increment Count(x), push $x_{count(x)}$ onto Stack(x), rename x to Stack(x)
- At use of x, rename x to Stack(x)
- When defs go out of scope, pop them off Stack(x)

Actual renaming algorithm

Algorithm 3 Classical SSA renaming algorithm

for all V : variables in original program **do**
    $Count(V) \leftarrow 0$
    $S(V) \leftarrow$ `EmptyStack`
end for
call Search(`EntryNode`)

```
procedure Search(X : BasicBlock)
for all A : statement in X do
    if A is not a φ-function then
        for all u : variables used in A do
            replace use of u with S(u) in A
        end for
    end if
    for all v : variables defined in A do
        i ← Count(v)
        replace definition of v with v_i in A
        push v_i onto S(v)
        Count(v) ← i + 1
    end for
end for
for all Y ∈ Succ(X) do
    let j be the index of the φ-function operands in Y that correspond to basic block
    X
    for all F: φ-function in Y do
        let V be the jth operand in F
        replace V with S(V) at the jth operand in F
    end for
end for
for all Z ∈ Children(X) do
    call Search(Z)
end for
for all A : statements in X do
    for all V_i : variables defined in A do
        let V be the original variable corresponding to V_i
        pop S(V)
    end for
end for
end procedure
```

procedure Search(X : BasicBlock)
for all A : statement in X **do**
  if A is not a ϕ-function **then**
    for all u : variables used in A **do**
      replace use of u with $S(u)$ in A
    end for
  end if
  for all v : variables defined in A **do**
    $i \leftarrow Count(v)$
    replace definition of v with v_i in A
    push v_i onto $S(v)$
    $Count(v) \leftarrow i + 1$
  end for
end for
for all $Y \in$ Succ(X) **do**
  let j be the index of the ϕ-function operands in Y that correspond to basic block X
  for all F: ϕ-function in Y **do**
    let V be the jth operand in F
    replace V with $S(V)$ at the jth operand in F
  end for
end for
for all $Z \in$ Children(X) **do**
  call Search(Z)
end for
for all A : statements in X **do**
  for all V_i : variables defined in A **do**
    let V be the original variable corresponding to V_i
    pop $S(V)$
  end for
end for
end procedure

synchronize Count

doall

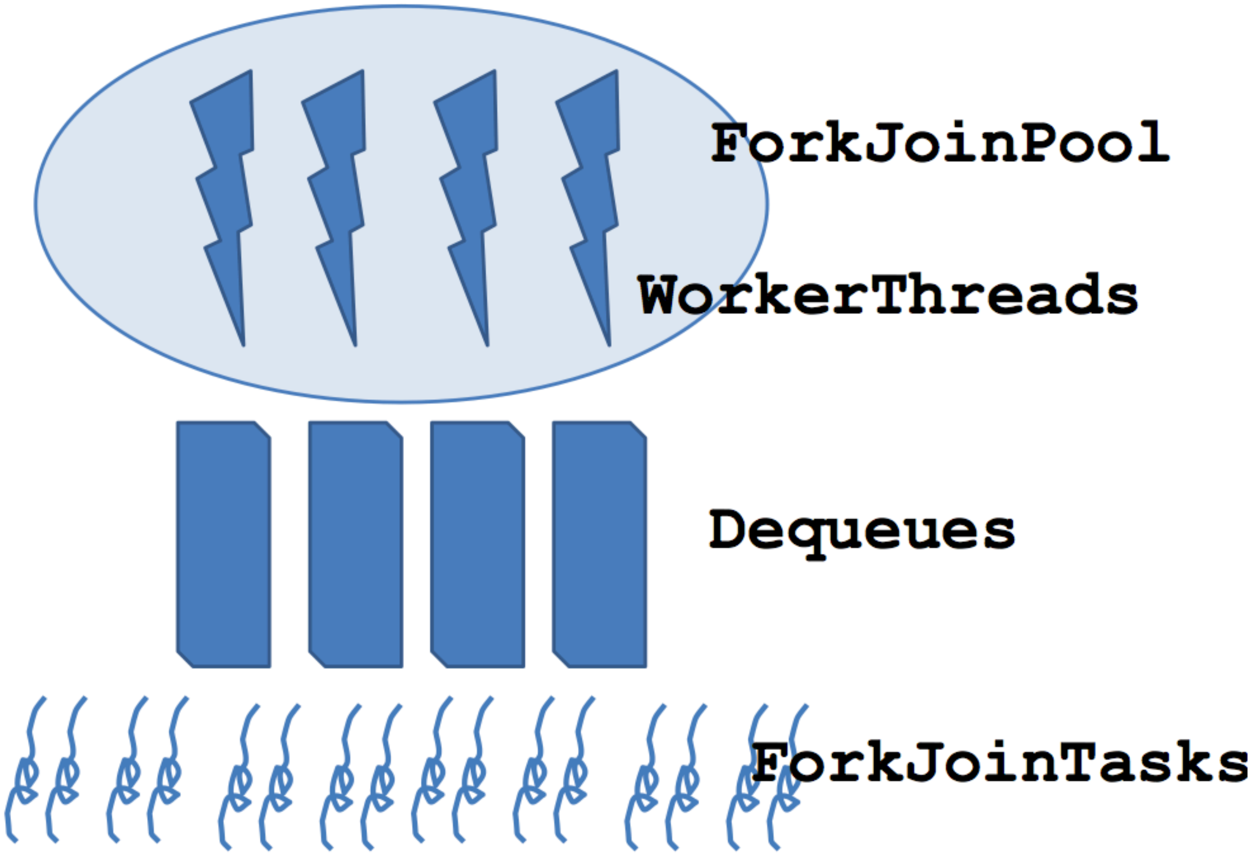
privatize S

enddo

Implementation Details

- Algorithms implemented in Soot
  - a Java bytecode compiler framework
- Parallelism via Java fork/join framework
  - thread pool
  - lightweight tasks
  - work-stealing queues
- Thread-safe data structures
  - **`java.util.concurrent.ConcurrentHashMap`**

jsr166y



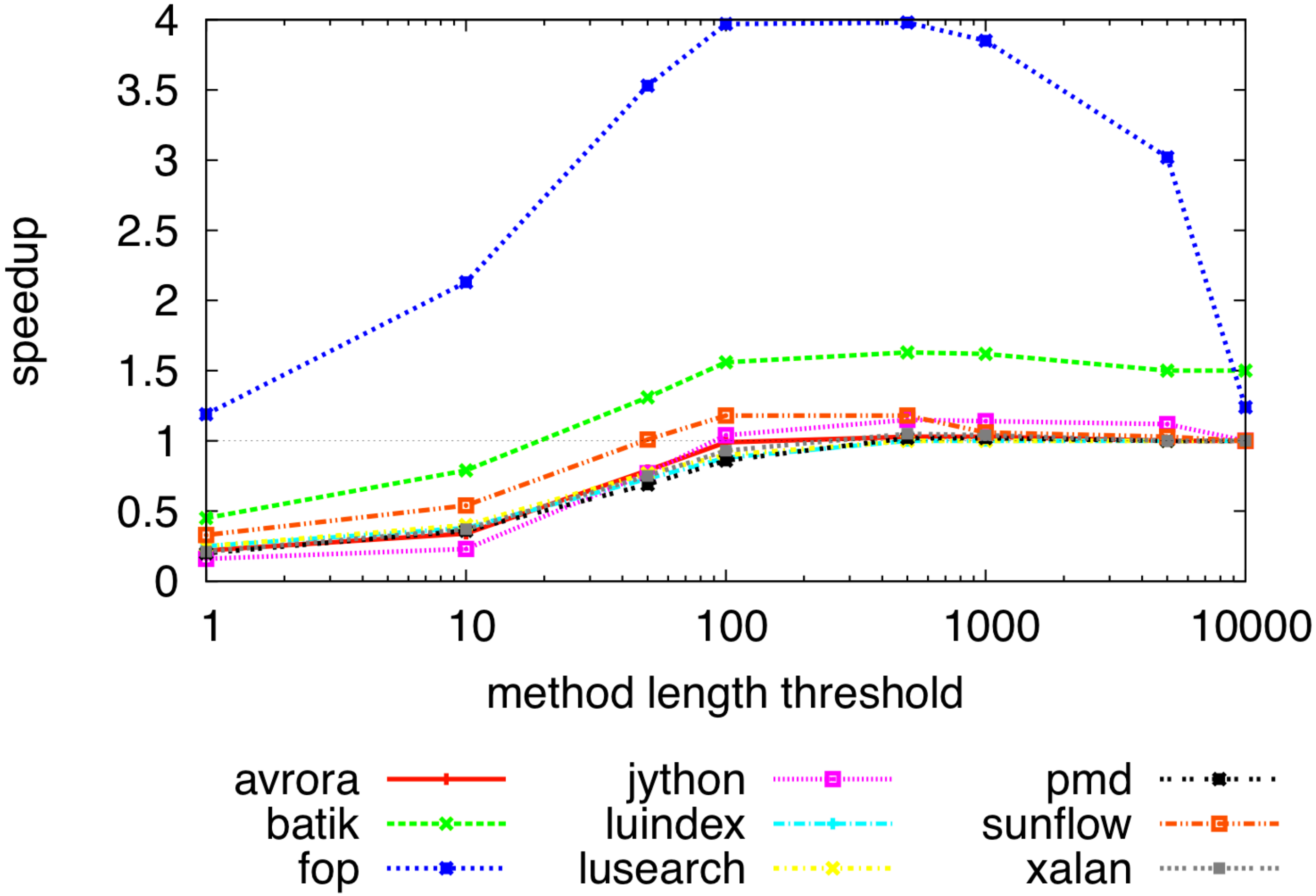
Evaluation

- Use standard Java benchmark programs
  - DaCapo, Java Grande
- Problem – some methods are so small that the parallel algorithm performs worse that the sequential one
- Solution – have a method size threshold, below which we always use sequential algorithm, above which we use parallel

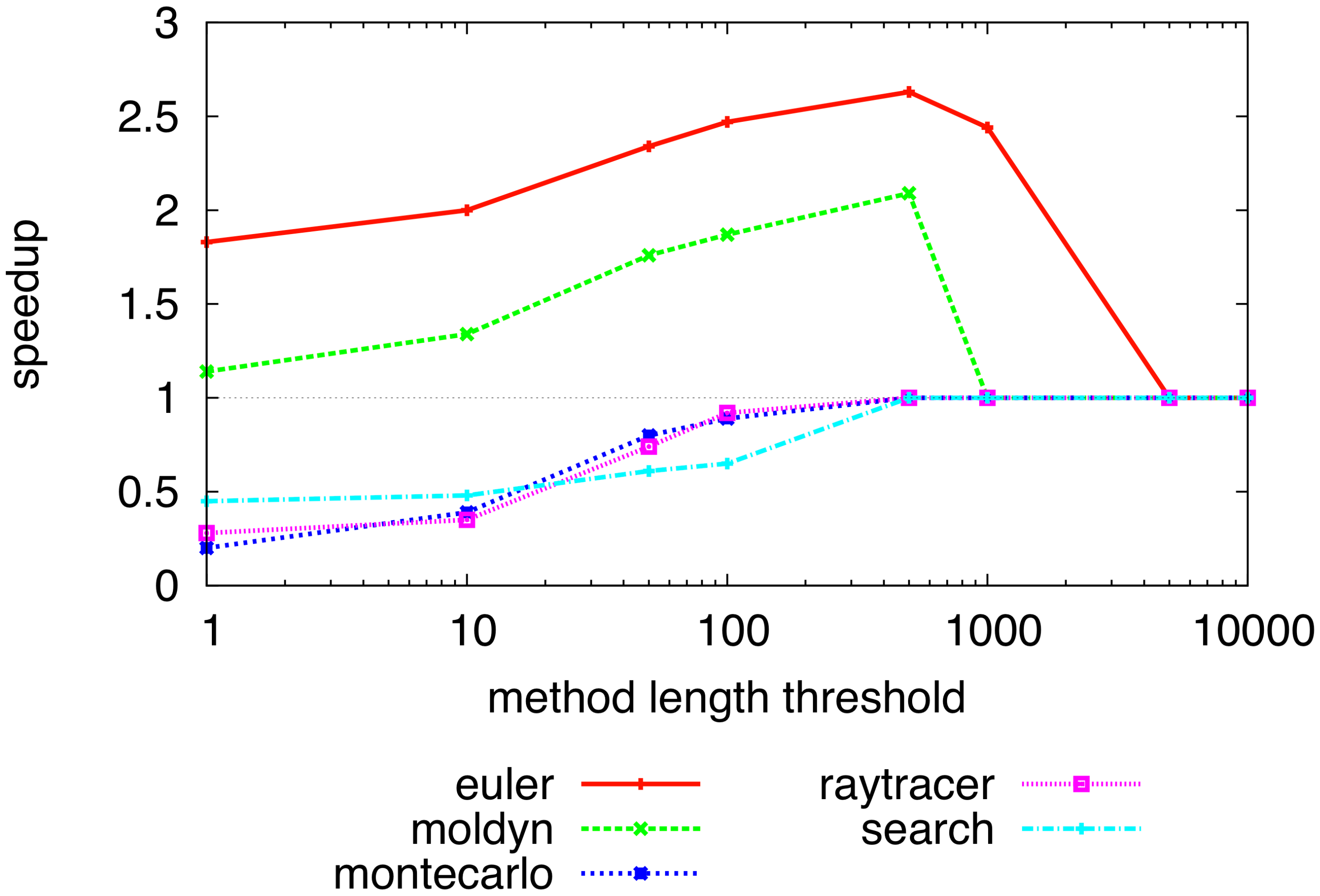
Evaluation Platform

- Core i7-920
  - 4 cores x 2 contexts
- JVM – 1.6, Hotspot v14.0-b16
- Soot – v2.4.0
- Linux – x86_64 v2.6.31

SSA Speedup on DaCapo



SSA Speedup on Java Grande



Explain with Method Size Stats

	app	*description*	*# methods*	*mean length*	*max length*
DaCapo	avrora	program simulator	2836	21.9	1083
	batik	SVG image processing	7137	34.3	41033
	fop	PDF generator	6749	44.5	33089
	jython	Python interpreter	20664	25.4	7846
	luindex	text indexing	1885	30.6	493
	lusearch	text search	1613	26.9	1187
	pmd	static analysis	6477	33.6	2881
	sunflow	raytracer	1109	50.4	6308
	xalan	XML parser	6189	29.5	2881
Java Grande	euler	fluid dynamics	27	295.81	1822
	moldyn	molecular dynamics simulation	20	102.20	931
	montecarlo	Monte Carlo simulation	178	17.65	211
	raytracer	raytracer	65	30.63	229
	search	alpha-beta search	29	86.34	465

http://www.docjar.com/html/api/org/apache/fop/text/linebreak/LineBreakUtils.java.html

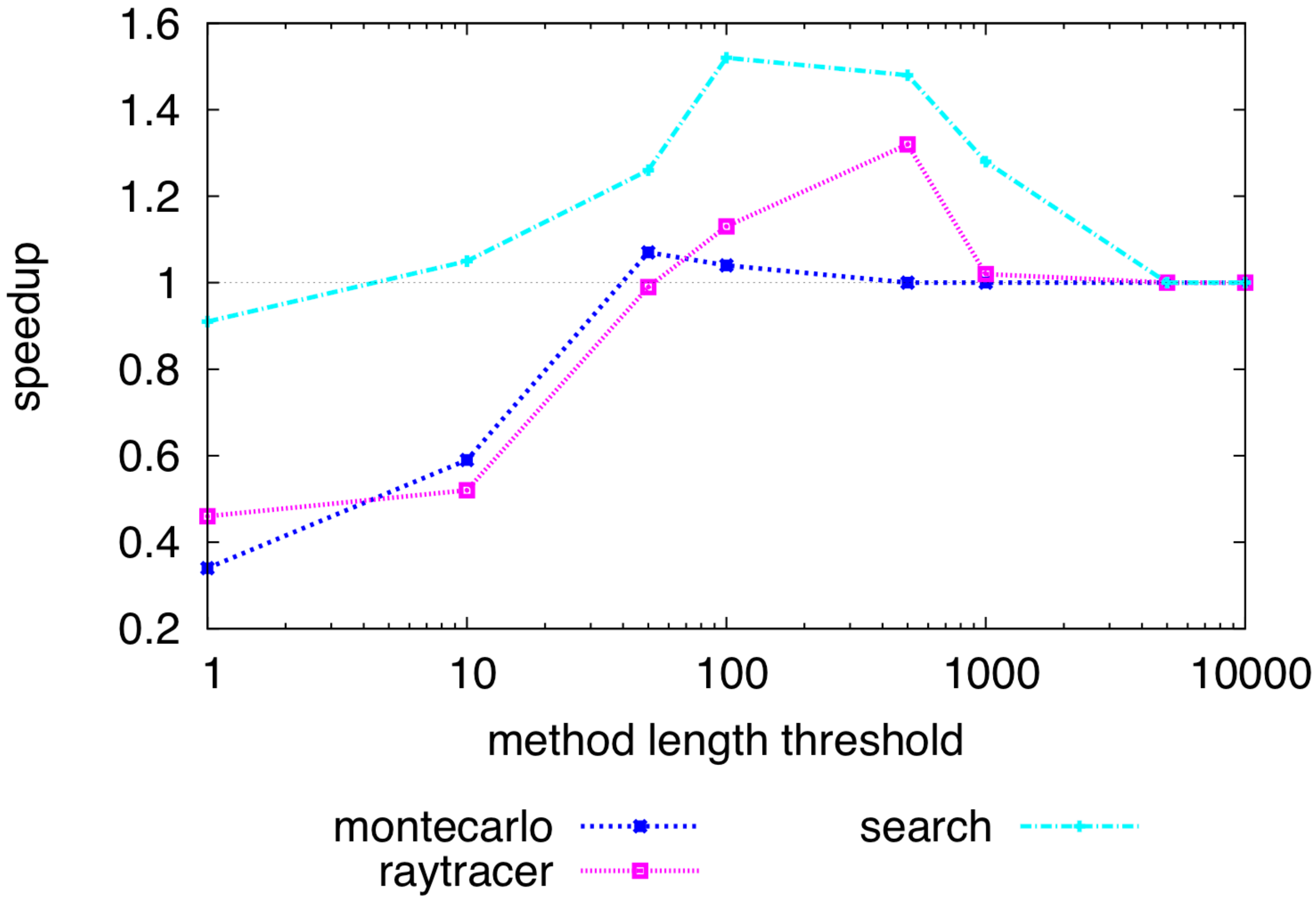
Google

dl Google Apple Wikipedia News (410) confs SE3

```
    private static byte lineBreakProperties[][] = new byte[512][];

    private static void init_0() {
        lineBreakProperties[0] = new byte[] { 9,9,9,9,9,9,9,9,9,4,22,6,6,10,9,9,9,9,9,9,
        lineBreakProperties[1] = new byte[] { 9,9,9,9,9,23,9,9,9,9,9,9,9,9,9,9,9,9,9,9,
        lineBreakProperties[2] = new byte[] { 2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
        lineBreakProperties[5] = new byte[] { 2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
        lineBreakProperties[6] = new byte[] { 9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,
        lineBreakProperties[7] = new byte[] { 0,0,0,0,2,2,2,2,2,2,2,0,2,0,2,2,2,2,2,2,
        lineBreakProperties[9] = new byte[] { 2,2,2,9,9,9,9,0,9,9,2,2,2,2,2,2,2,2,2,2,
        lineBreakProperties[10] = new byte[] { 2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
        lineBreakProperties[11] = new byte[] { 2,2,2,2,2,2,2,2,0,18,4,0,0,0,0,0,0,9,9,9,
        lineBreakProperties[12] = new byte[] { 2,2,2,2,0,0,0,0,0,0,0,27,11,18,2,2,9,9,9,
        lineBreakProperties[13] = new byte[] { 2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
        lineBreakProperties[14] = new byte[] { 2,2,2,2,2,2,2,2,2,2,2,2,2,2,0,2,2,9,2,2,
        lineBreakProperties[15] = new byte[] { 2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
        lineBreakProperties[16] = new byte[] { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
        lineBreakProperties[18] = new byte[] { 0,9,9,9,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
        lineBreakProperties[19] = new byte[] { 0,9,9,9,0,2,2,2,2,2,2,2,2,0,0,2,2,0,0,2,
        lineBreakProperties[20] = new byte[] { 0,9,9,9,0,2,2,2,2,2,2,0,0,0,0,2,2,0,0,2,
        lineBreakProperties[21] = new byte[] { 0,9,9,9,0,2,2,2,2,2,2,2,2,2,0,2,2,2,0,2,
        lineBreakProperties[22] = new byte[] { 0,9,9,9,0,2,2,2,2,2,2,2,2,0,0,2,2,0,0,2,
        lineBreakProperties[23] = new byte[] { 0,0,9,2,0,2,2,2,2,2,2,0,0,0,2,2,2,0,2,2,
```

Effects of Method Inlining



Throw rotten fruit now

- Most methods are too short for parallel algorithm.
- SSA construction time is insignificant in overall compilation process.
- Why not parallelize SSA construction for multiple methods at once?

Concluding Remarks

- We have presented one technique for parallelization of data flow analysis, to take advantage of multicore resources.

- We see overhead of fork/join parallelism versus saving of parallel execution – need to find threshold.