

Introduction to Habanero Java

David Bunde, Jaime Spacco, Casey Samoore

Knox College

Acknowledgements

- Material drawn from a tutorial created with contributions from Johnathan Ebbers, Maxwell Galloway-Carson, Michael Graf, Sung Joo Lee, and Andrei Papancea
- Work partially supported by NSF DUE-1044299. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation



Schedule

- Introduction
- Core features
- Hands-on session
- Break
- “Other” features
- Teaching experiences

Rationale for parallelism

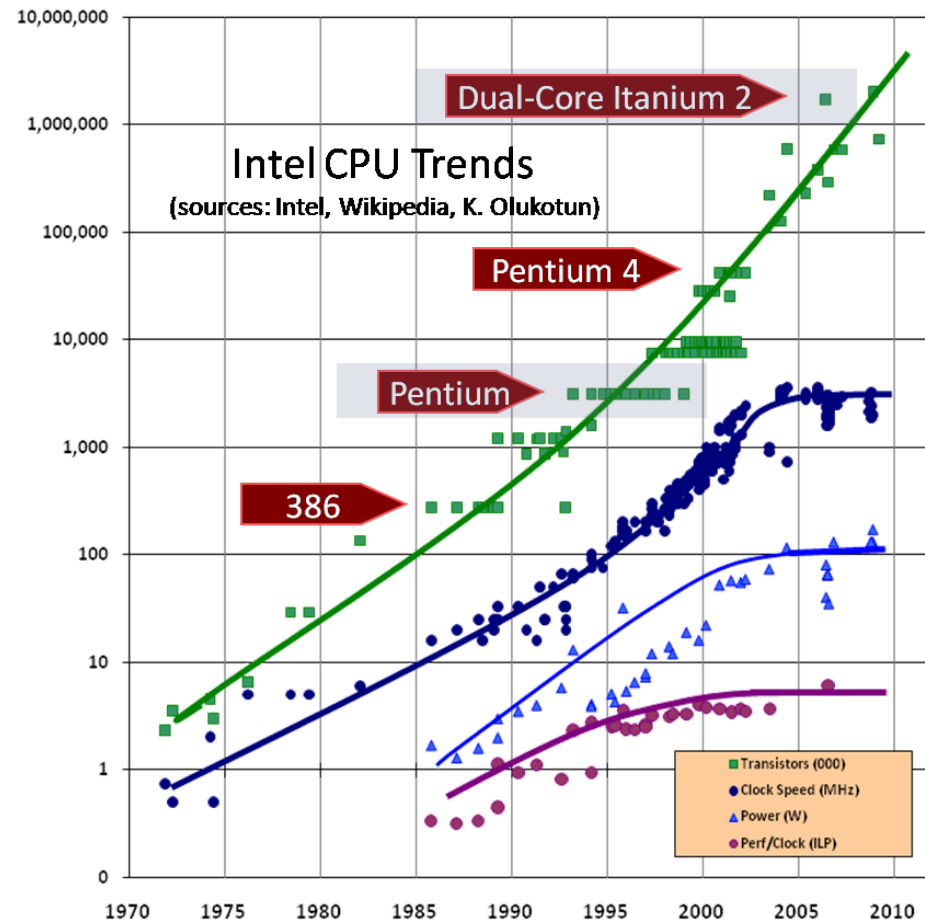


Figure: Herb Sutter "The free lunch is over: A fundamental turn toward concurrency in software"

Dr. Dobb's Journal, 30(3), March 2005.

<http://www.gotw.ca/publications/concurrency-ddj.htm>

Basic Facts about Habanero Java (HJ)

- Under development in the group of Vivek Sarkar (Rice Univ.)
- Addition of small number of keywords to Java
 - Designed for teaching and research on parallel technology
- “Pedagogic extension” of X10, a parallel language coming out of DARPA’s High Productivity Computing Systems program
- Programs run on the Java virtual machine

Why HJ?

- Easy to introduce if students already use Java
 - We did a short introduction in CS 2
- Allows quicker expression of parallel algorithms than traditional languages
 - Facilitates prototyping of alternatives
 - Lets students focus on the algorithm and not the code

Simplified Parallel Programming

- Toy application: Counting prime numbers

```
int pCount = 1;    // # primes found (starting w/ 2)
```

```
for(nextCand = 3; nextCand < 2000000; nextCand += 2)  
    if(isPrime(nextCand))  
        pCount++;
```

Java Threads Version (Part 1)

```
class PrimeFinder implements Runnable {  
    ...  
    public void run() {  
        int pCount = 0;  
  
        for(long nextCand = from; nextCand < to; nextCand +=2)  
            if(isPrime(nextCand))  
                pCount++;  
  
        synchronized(lock) { sharedPCount += pCount; }  
    }  
}
```


Java Threads Version (Part 2)

```
sharedPCount = 1;  
PrimeFinder p1 = new PrimeFinder(3, 1000000);  
Thread t1 = new Thread(p1);  
PrimeFinder p2 = new PrimeFinder(1000001, 2000000);  
Thread t2 = new Thread(p2);  
  
t1.start();  
t2.start();  
  
t1.join();  
t2.join();
```

HJ Version (Part 1)

```
public void count(long from, long to) {  
    int pCount = 0;  
  
    for(long nextCand = from; nextCand < to; nextCand +=2)  
        if(isPrime(nextCand))  
            pCount++;  
  
    isolated { sharedPCount += pCount; }  
}
```

HJ Version (Part 2)

```
sharedPCount = 1; //(starting with 2)
```

```
finish {
```

```
    async count(3, 1000000);
```

```
    count(1000001, 2000000);
```

```
}
```

Your Presenters are ...

- Interested in high-level approaches to parallel programming
- Learning HJ and beginning to use HJ with our students
- **NOT connected to the developers of HJ**

HJ Resources

- Downloading and installation instructions:
<https://wiki.rice.edu/confluence/display/PARPROG/HJDownload>
- Materials for this workshop:
<http://faculty.knox.edu/dbunde/teaching/hj/CCSC-MW/>
- Our HJ tutorial:
<http://faculty.knox.edu/dbunde/teaching/hj/>
- Official HJ webpage:
<https://wiki.rice.edu/confluence/display/HABANERO/HJ>
- Nice paper summarizing HJ:
Habanero-Java: the New Adventures of Old X10
<http://www.cs.rice.edu/~vsarkar/PDF/hj-pppj11.pdf>

Core Features

Compiling and Running

- Use editor to create HJ program (Hello.hj):

```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

- Compile: `hjc Hello.hj` (creates Hello.class)
- Run: `hj Hello`

forall Loops

- Self-contained parallel loop

```
forall(point [i] : [0:9]) {  
    //loop body  
}
```

- Automatically splits iteration space into tasks

Points

- Store integer-valued location in Cartesian space
- point p supports:
 - $p.rank$ gives dimensionality of p
 - $p.get(i)$ gives i^{th} coordinate of p
 - lexicographic comparison: $p.lt(other)$, $p.le(other)$,
 $p.gt(other)$, $p.ge(other)$

Regions

- Sets of points formed as product of ranges
 - $[1:3]$ contains 1D points $[1], [2], [3]$
 - $[1:3,1:3]$ contains 2D points $[1,1], [1,2], [1,3],$
 $[2,1], [2,2], \dots$
 - and so on
- Regions are always rectilinear volumes

Implementation of forall

- Iteration corresponding to each point becomes a task
- Tasks are completed asynchronously by a team of threads
- Execution proceeds when last task completes

Simple Error

- Code to increment `val` 20,000 times:

```
forall(point [i] : [0:19999])  
    val++;  
System.out.println(val);
```

- Actually increments fewer (varies by run)

Eliminating Race Conditions

- Need to mark critical section with `isolated`
`forall(point [i] : [0:19999])`
`isolated {`
`val++;`
`}`

Implementation of isolated

- HJ promises that no pair of conflicting isolated blocks will run concurrently
 - conflicting means both access some variable and at least one is a write
- Allows transaction-based implementation
- Currently just a single global lock

How should isolated be used to correct this code?

```
public static void main(String[] args) {  
A:     int num = 0;  
B:     forall(point [p] : [2:10000])  
C:         if(isPrime(p)) {  
D:             num = num + 1;  
        }  
E:     System.out.println(num);  
}
```

Which block of code should be surrounded with the isolated keyword in order to ensure that this code prints the correct result?

- A:
- B:
- C:
- D:
- E:

Creating Asynchronous Tasks

- Can also create asynchronous tasks by hand:

```
async System.out.println("Hello");  
async System.out.println("World!");  
async System.out.println("Welcome");  
async System.out.println("to");  
async System.out.println("HJ!");
```

- Each println runs as separate task
- Body of `async` can also be a block of code

What are the possible outcomes of the following program?

```
async {  
    System.out.println(10);  
    System.out.println(5);  
}  
System.out.println(3);
```

Which are possible outputs for this Habanero Java code snippet?

- A: 10, 5, 3
- B: 10, 3, 5
- C: 5, 10, 3
- D: A and B
- E: All of the above

What are the possible outcomes of the following program?

```
async {  
    System.out.println(10);  
    System.out.println(5);  
}  
System.out.println(3);
```

Which are possible outputs for this Habanero Java code snippet?

A: 10, 5, 3

B: 10, 3, 5

C: 5, 10, 3

D: A and B (2 of 9 students got this right)

E: All of the above

What are the possible outcomes of the following program?

```
async{
    System.out.println(10);
}
System.out.println(5);
async{
    System.out.println(3);
}
```

Which are possible outputs for this Habanero Java code snippet?

- A: 10, 5, 3
- B: 10, 3, 5
- C: 5, 10, 3
- D: A and B
- E: A and C

What are the possible outcomes of the following program?

```
async{
  System.out.println(10);
}
System.out.println(5);
async{
  System.out.println(3);
}
```

Which are possible outputs for this Habanero Java code snippet?

A: 10, 5, 3

B: 10, 3, 5

C: 5, 10, 3

D: A and B

E: A and C (1 of 9 students got this right)

Effect of async on Variable Scope

- Easy to get scope errors when using async
- Tasks can use
 - their own local variables
 - a private read-only copy of variables from outer scope
 - static variables

Waiting for tasks to complete

- Use finish to wait for a group of tasks:

```
finish {  
  async function1();  
  async function2();  
}
```

- Both function1 and function2 complete before execution proceeds beyond this block

Making “forall” by hand

```
finish {  
  for( loop condition ) {  
    async {  
      loop body  
    }  
  }  
}
```

What are the possible outcomes of the following program?

```
finish {  
    async {  
        System.out.println(10);  
    }  
    async {  
        System.out.println(5);  
    }  
    System.out.println(3);  
}
```

Which are possible outputs for this Habanero Java code snippet?

- A: 10, 5, 3
- B: 10, 3, 5
- C: 5, 10, 3
- D: A and B
- E: All of the above

What are the possible outcomes of the following program?

```
finish {  
    async {  
        System.out.println(10);  
    }  
    async {  
        System.out.println(5);  
    }  
    System.out.println(3);  
}
```

Which are possible outputs for this Habanero Java code snippet?

A: 10, 5, 3

B: 10, 3, 5

C: 5, 10, 3

D: A and B

E: All of the above (4 of 9 students got this right)

What are the possible outcomes of the following program?

```
finish {  
  async {  
    System.out.println(10);  
  }  
  async {  
    System.out.println(5);  
  }  
}  
System.out.println(3);
```

Which are possible outputs for this Habanero Java code snippet?

A: 10, 5, 3

B: 10, 3, 5

C: 5, 10, 3

D: A and B

E: A and C

What are the possible outcomes of the following program?

```
finish {  
  async {  
    System.out.println(10);  
  }  
  async {  
    System.out.println(5);  
  }  
}  
System.out.println(3);
```

Which are possible outputs for this Habanero Java code snippet?

A: 10, 5, 3

B: 10, 3, 5

C: 5, 10, 3

D: A and B

E: A and C (6 of 9 students got this right)

Complicated task structure

```
void countZero(int[] A, int from, int to) {
    if((to - from) < threshold) {
        //serially compute local count using for loop
        isolated { numZeros += localCount; }
    } else {
        int mid = from + (to - from)/2;
        async countZero(A, from, mid);    //1st half w/ new task
        countZero(A, mid, to);           //recurse for 2nd
    }
}
...
finish {
    countZero(array, 0, array.length);
} //waits at end of finish until all tasks complete
```

Recursive Task Organization

- If problem is small enough, task solves it
- Otherwise, spawn subtask for first half of it
- Prevents overhead from dominating the running time, but creates large number of tasks when there is enough work

Asynchronous Return Values

- What about asynchronous operations that return values?
 - Can't assign to a variable yet since value hasn't been calculated
 - Inelegant to have the function set a global or use other indirect return techniques

Futures

- Solution is a future
 - container to store the value
 - call immediately returns the container
 - provides accessor method (get) that blocks until calculation completes (no need for a finish)

Syntax of future

```
final future<T> var = {
```

```
    ...
```

```
    return retVal; //something of type T
```

```
};
```

```
...
```

```
T result = var.get(); //blocks if necessary
```


CountZero using future

```
public static int countZero(int[] array, int from, int to) {
    if((to - from) < threshold) {
        //serially compute local count using for loop
        return localCount;
    }
    int halfRange = (to - from)/2;
    final future<int> firstFuture = async<int> {
        return countZero(array, from, from+halfRange);
    };
    int second = countZero(array, from+halfRange, to);
    return firstFuture.get() + second;
}
...
countZero(array, 0, array.length);
```

What HJ keyword will help us fix the error?

```
public static void main(String[] args) {  
    int num = 0;  
    forall(point [p] : [2:10000])  
        if(isPrime(p)) {  
            num = num + 1;  
        }  
    System.out.println(num);  
}
```

This code sometimes returns the wrong result. What HJ keyword will help us fix the error?

- A: async
- B: future
- C: finish
- D: isolated
- E: replace the forall loop with a formost loop

What HJ keyword will help us fix the error?

```
public static void main(String[] args) {  
    int num = 0;  
    forall(point [p] : [2:10000])  
        if(isPrime(p)) {  
            num = num + 1;  
        }  
    System.out.println(num);  
}
```

This code sometimes returns the wrong result. What HJ keyword will help us fix the error?

A: async

B: future

C: finish

D: isolated

E: replace the forall loop with a formost loop

Hands-on Session

Teaser Trailer: Other Features

Some Other Features

- Other Synchronization: Barriers and Phasers
- Notation for locality/affinity: Places
- Ability to access array contents using different indexing schemes: Array Views
- Complex Numbers

Teaching with HJ

Teaching with HJ

- Quick introduction to parallelism in CS 2
- Planned use in parallel programming course
- Other possible uses

Quick Intro. to Parallelism in CS 2

- Goal was to expose students to parallel concepts via HJ
- Jaime introduced n-queens, guest lecture by Casey, and then two days by Jaime
- Introduction on parallelism and worked through online tutorial together
- Assessment via online quiz (content and attitudinal questions)

Why CS 2?

- Introduce parallelism throughout our major
 - Wanted early introduction
- CS 2 is “gateway” course; pre-requisite for “Core courses”, which are taken in any order
- Students have shown interest in CS
- Students have a decent understanding of basic computer science ideas and Java syntax

Why I (Casey) am here

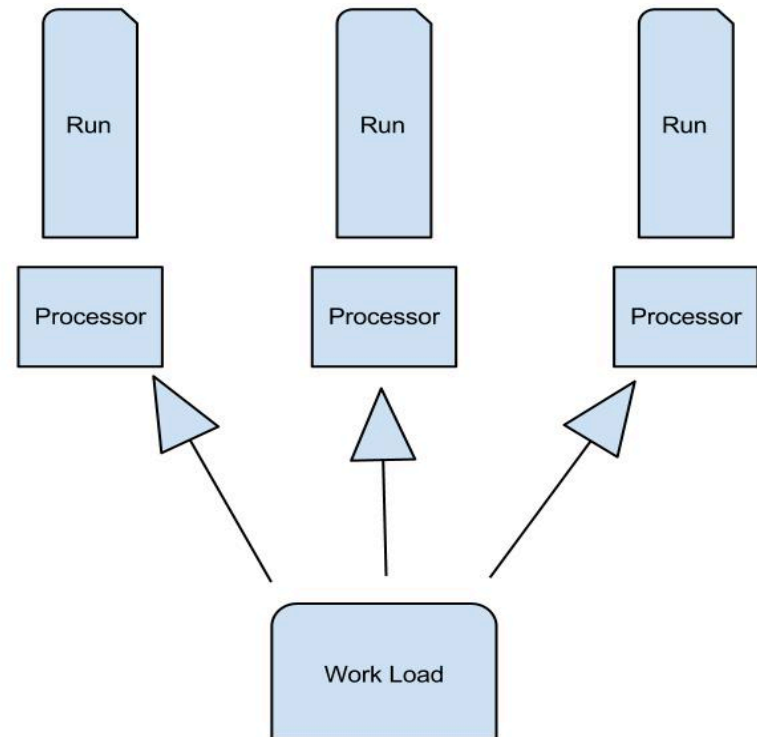
- During the summer of 2011, I was one of three students hired to research parallel programming languages
- Quickly learned HJ, then wrote exercises and the online tutorial

Guest Lecture

- The lecture itself was composed of 3 parts
 - What is parallel programming?
 - What are common aspects of parallel programming?
 - How does HJ deal with these aspects?

What is Parallel Programming?

- What does it mean for something to run in parallel?
- How do computers do this?
- Why is it important?



Aspects of Parallel Programming

- Race Conditions
 - Bank Example
- Deadlock
- Threads
- Joining/Forking
- Load Balancing



Image from <http://www.glommer.net/blogs/?p=189>

HJ Coverage

- Couldn't teach the whole language in a short unit
- Focused on the basic features, as related to the parallel concepts
 - async
 - forall
 - isolated
 - finish

Tutorial Walkthrough

What Students Learned

- async: Most students made the connection between async and creating a thread
 - However, they struggled to apply this to identify possible interleavings
- isolated: A bit shakier, but students understood that it prevented threads from interfering with each other
- finish: Students knew that the program would have to wait until all processes reached a certain point
- forall: students explained it as the parallel version of a for loop, but did not demonstrate an understanding of how it relates to threads (need to revise the question)

Student Impressions

- Students universally agreed that parallel programming was an important topic
- Most thought the best part of the unit was seeing exciting speed up and how easy it was to achieve
- The main criticism of the unit was that it was too short and they didn't have enough time to experiment and learn HJ

Our Impressions

- Brief unit will not give students mastery of HJ
- It did seem to help them grasp major parallel programming ideas
- Students were excited to do parallel programming
- HJ made it easy to demonstrate parallelism, even in CS 2

HJ in Parallel Programming (W `13)

- Project to parallelize supercomputer simulator
 - “Large” code base (138 files, ~13K LOC)
 - Long-running simulations (sometimes weeks+)
 - Several types of exposed parallelism
- They will try to identify/measure available parallelism, then exploit it

How else might you use HJ?

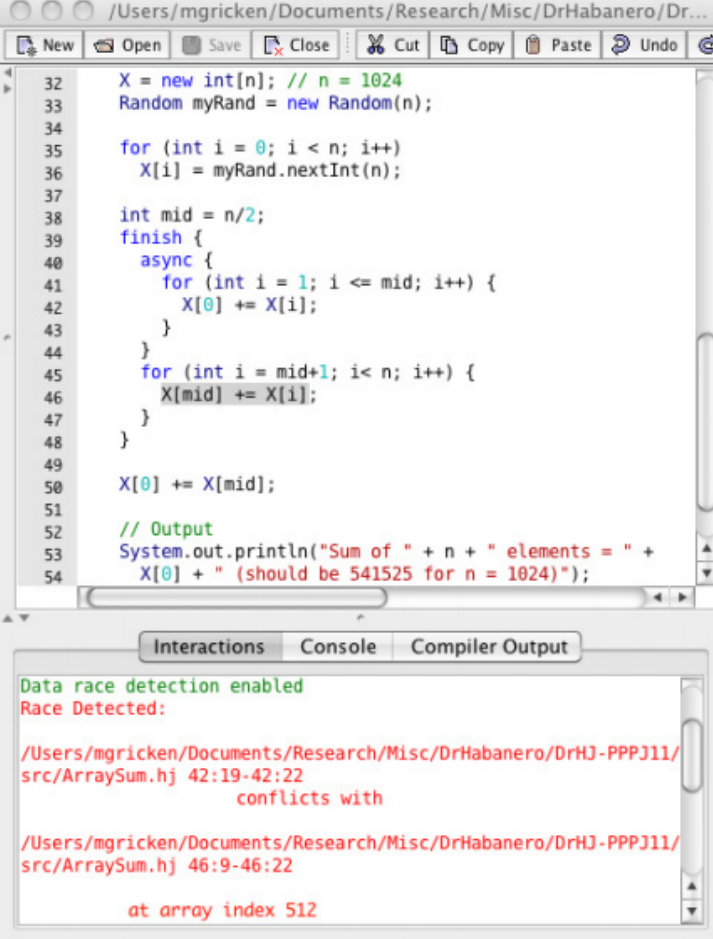
- Parallel Computing
 - quick prototyping of parallel design alternatives
- Software Design
- Operating Systems
 - high-level counterpart to threads
- Artificial Intelligence
 - (or other courses w/ computationally-intense projects)
- Independent Projects

Drawbacks with HJ

- Research project, not finished
 - Doesn't support newest Java (no generics until newest release; still no foreach loops or reflection)
 - Terse and confusing error messages
- No integrated development environment (IDE)
- Limited documentation, no textbook

DrHJ: Prototype IDE

- Dr HJ is an IDE designed for Habanero Java
- Students had already used Eclipse so Dr HJ would possibly be a familiar working environment
- Not as advanced as Eclipse, but we hoped it would be more familiar



```
32 X = new int[n]; // n = 1024
33 Random myRand = new Random(n);
34
35 for (int i = 0; i < n; i++)
36     X[i] = myRand.nextInt(n);
37
38 int mid = n/2;
39 finish {
40     async {
41         for (int i = 1; i <= mid; i++) {
42             X[0] += X[i];
43         }
44     }
45     for (int i = mid+1; i < n; i++) {
46         X[mid] += X[i];
47     }
48 }
49
50 X[0] += X[mid];
51
52 // Output
53 System.out.println("Sum of " + n + " elements = " +
54     X[0] + " (should be 541525 for n = 1024)");
```

Interactions Console Compiler Output

```
Data race detection enabled
Race Detected:

/Users/mgricken/Documents/Research/Misc/DrHabanero/DrHJ-PPPJ11/
src/ArraySum.hj 42:19-42:22
conflicts with

/Users/mgricken/Documents/Research/Misc/DrHabanero/DrHJ-PPPJ11/
src/ArraySum.hj 46:9-46:22

at array index 512
```

But

- Dr HJ was not stable on our (Windows) lab machines at the time of the lecture
 - It frequently crashed
 - It would not work on certain machines
- Eventually we decided not to use it
- Hopefully, it will become more reliable in the future, making HJ more accessible for students accustomed to IDEs

Conclusions

- Very appealing that it's based on Java
 - Low introduction cost and can selectively teach desired features
 - Doesn't support all of Java, but has main features
- Currently should be taught using command-line tools
- Useful to illustrate parallel concepts
 - Even with brief exposure, it helps make parallel ideas concrete

Your Feedback

- What are your impressions of HJ?
- How likely are you to adopt it?
 - What course(s) will you use it in?
- What resources would help you adopt it?