Computing the Quadrillionth Digit of π

A Supercomputer in the Garage

Ed.Karrels@gmail.com

Overview

- Computing a few digits of π
- Implementing in CUDA
- Performance results
- Production challenges

Breaking the Record

- Old record (2010): 2 quadrillionth bit ($2*10^{15}$)
  - Yahoo Hadoop 1000 node cluster, 23 days
  - 4-GPU CUDA box, 24 days
- 4 quadrillionth bit
  - 37 days on one 4-GPU box
- 8 quadrillionth bit
  - 26 days on 26 boxes, 30 GPUs

Computing π

- Irrational (digits don't repeat)
- Transcendental (infinite-length formula)
  - Usually an infinite sum

$$\sum_{i=0}^{inf} \ldots$$

Sample π formulas

$$\frac{\pi}{2} = 1 + \frac{1}{3} + \frac{1 \cdot 2}{3 \cdot 5} + \frac{1 \cdot 2 \cdot 3}{3 \cdot 5 \cdot 7} + \frac{1 \cdot 2 \cdot 3 \cdot 4}{3 \cdot 5 \cdot 7 \cdot 9} \cdots$$

$$\arctan\left(\frac{1}{x}\right) = \frac{1}{x} - \frac{1}{3x^3} + \frac{1}{5x^5} - \frac{1}{7x^7} + \ldots$$

$$\frac{\pi}{4} = \arctan(1) = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \ldots$$

$$\frac{\pi}{4} = 4\arctan\left(\frac{1}{5}\right) - \arctan\left(\frac{1}{239}\right)$$

Chudnovsky

$$\frac{1}{\pi} = 12 \sum_{k=0}^{inf} \frac{(-1)^k (6\mathrm{k})!\,(13591409 + 545140134\mathrm{k})}{(3\mathrm{k})!\,k!^3\,640320^{(3\mathrm{k}+3/2)}}$$

Computing the first N digits of π

$$\sum_{i=0}^{f(N)} \ldots$$

```
i=0  3.1333333333333333
i=1  0.00808913308913309
i=2  0.000164923924115 10
i=3  0.00000506722085386
i=4  0.00000018789290094
i=5  0.00000000776775122
...
Sum: 3.141592653228088
```

- Compute each term to N digits
- Iterate until terms are small enough ($< 10^{-N}$)
  - Chudnovsky formula: ~13 digits per iteration
  - Limiting factor: disk speed (assuming fast multiply)

Computing the N^{th} digit of π

```
10^7 * 3.14159265358979323846264338327…
     = 31415926.535897932384626433832…
…drop the integer part…
                .535897932384626433832…
```

- Scale the calculation by 10^N
- Drop everything left of the decimal
- Want the billionth digit?
  - Simply (!) multiply by $10^{1,000,000,000}$
- Must be an easier way…

Dropping upper digits

- What's the last digit of 1234567 * 9876543?
  - Only need the last digits of the factors: 7 and 3

  (7 * 3) = 21

  (7 * 3) % 10 = 1

  (a*b)%m == ((a%m) * (b%m)) % m
- If result will be modulo-m, do the modulo beforehand

Computing large powers

- 3^{53} 32 16 4 1

  $53 = 1\ 1\ 0\ 1\ 0\ 1_2$

  $3^{53} = 3^{32} * 3^{16} * 3^4 * 3^1$

  X^n in O(log n)

- 3^{53} % m = [(3^{32} % m)*(3^{16} % m)*(3^4 % m)*(3^1 % m)] % m
- (X^n) % m very efficient
  - "Modular exponentiation"
  - Modulo m at each step to keep numbers small
  - Heavily used in crypto

Bailey-Borwein-Plouffe Formula

$$\pi = \sum_{k=0}^{inf} \left[\frac{1}{16^k} \left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right) \right]$$

- Infinite sum (like most π formulas)
- Each term multiplied by 16^k
- What if we want the N^{th} digit?

Nth digit, BBP formula

$$10^N \frac{1}{16^k} \cdots$$ Not very interesting

How about base 16?

$$16^N \frac{1}{16^k} = \frac{16^N}{16^k} = 16^{N-k}$$

Much more interesting…

One term of BBP:

$$16^N \frac{1}{16^k} \frac{4}{8k+1} = \frac{16^{N-k} * 4}{8k+1} = \frac{2^{4N-4k+2}}{8k+1}$$

Use modulo to drop integer part

$$\frac{2^{4N-4k+2}}{8k+1}$$

$$fract\left(\frac{17}{7}\right) = fract(2\frac{3}{7}) = \frac{3}{7}$$

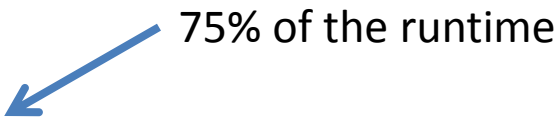
That looks familiar!

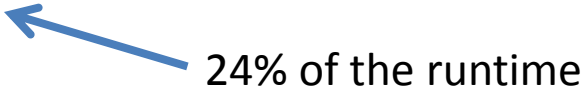
$$fract\left(\frac{2^P}{M}\right) = \frac{2^P \% M}{M}$$

Main loop

```
sum = 0;
END = …function of target digit…
for (k = 0; k < END; k++) {
  P = …function of k…
  M = …function of k…

  t = modpow(2, P, M);
  sum += t / M;
}
```





- Each term is independent
- Combine results with simple summation

CUDA Implementation

- Embarrassingly parallel
- Very little data movement
- Hard part: 128-bit and 192-bit integer types
  - 64-bit modpow: need 128-bit when squaring
  - Division: 192bit / 64 bit → 128 bit
    - Start with estimate, Newton-Raphson to refine
  - Multiprecision arithmetic: need the carry bit
  - Lots of inline PTX assembly

Optimizations

- Upgrade from NVCC from 4 to 5: 3x
- Bellard formula: +40%
- Montgomery Reduction: 7x

Bellard formula

$$\pi = \frac{1}{64}\sum_{k=0}^{inf}\left[\frac{(-1)^k}{1024^k}\left(\frac{256}{10k+1} - \frac{64}{10k+3} - \frac{4}{10k+5} - \frac{4}{10k+7} + \frac{1}{10k+9} - \frac{32}{4k+1} + \frac{1}{4k+3}\right)\right]$$

16 → 1024
4 bits → 10 bits

All odd denominators

- 2^P % M much easier if M is odd
- Montgomery Reduction: use 2^{-1} mod M, change modular base from M to a power of 2
- Modulo power of 2: bitwise ‘and’
- Bonus: alternating sign minimizes round-off error

Performance

- O(N log D) time, O(D) space
  - N=which digit, D=length of result
- Time to compute 100 millionth digit:
  - GPU (GTX 680): 1.57 seconds
  - CPU (3.4GHz i5): 211 seconds
    - Using GNU GMP math library for large integers
- Not a fair comparison
  - CPU code is a single thread, not vectorized, not hand-optimized

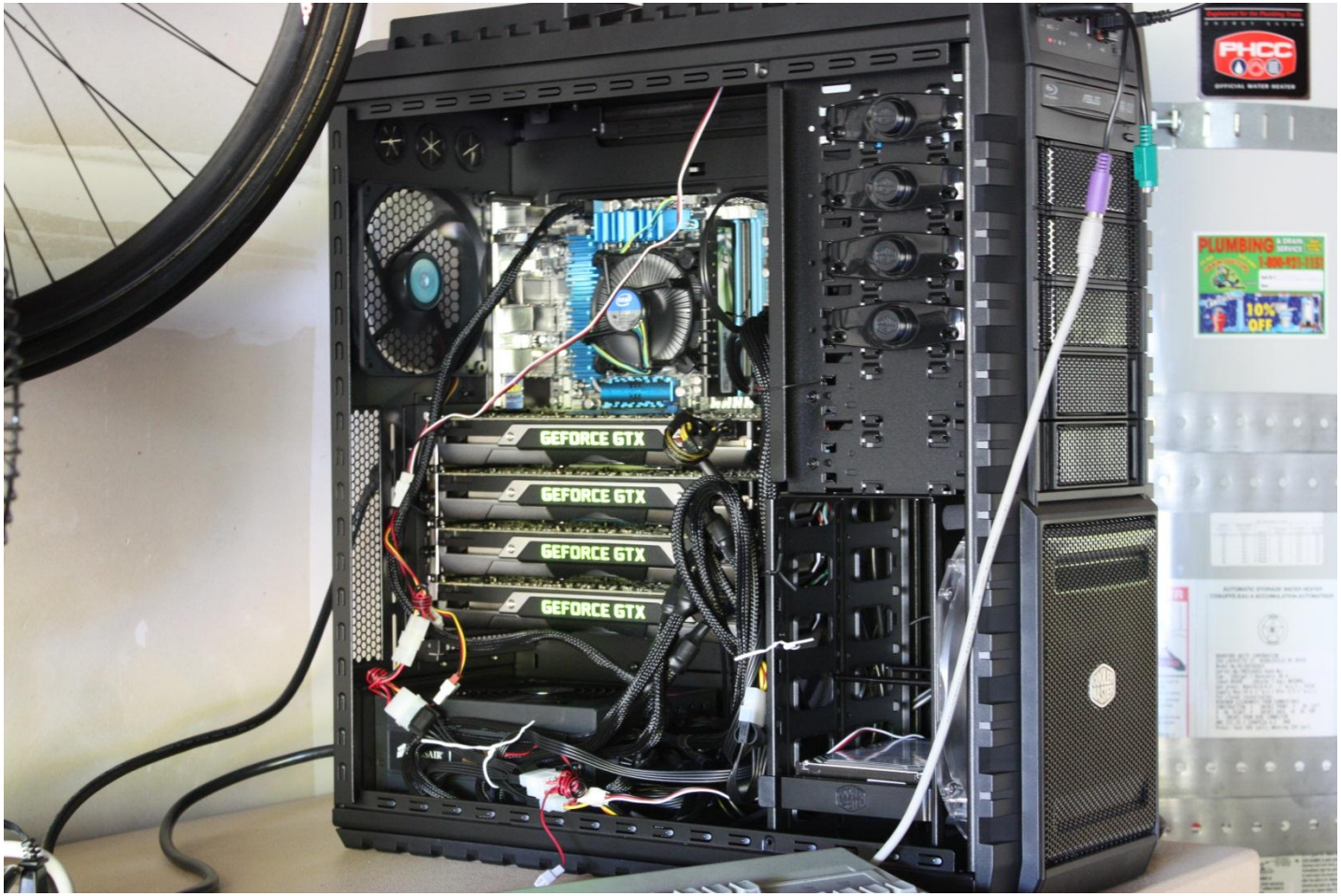
Performance

- Better comparison: Yahoo Hadoop cluster
- Time to compute 500 trillionth hex digit:
  - Me: one 4-GPU machine
    - 24 days
    - Hardware costs: $5000
  - Yahoo: 1000 8-core machines
    - 23 days
    - Hardware costs: $500,000-ish?

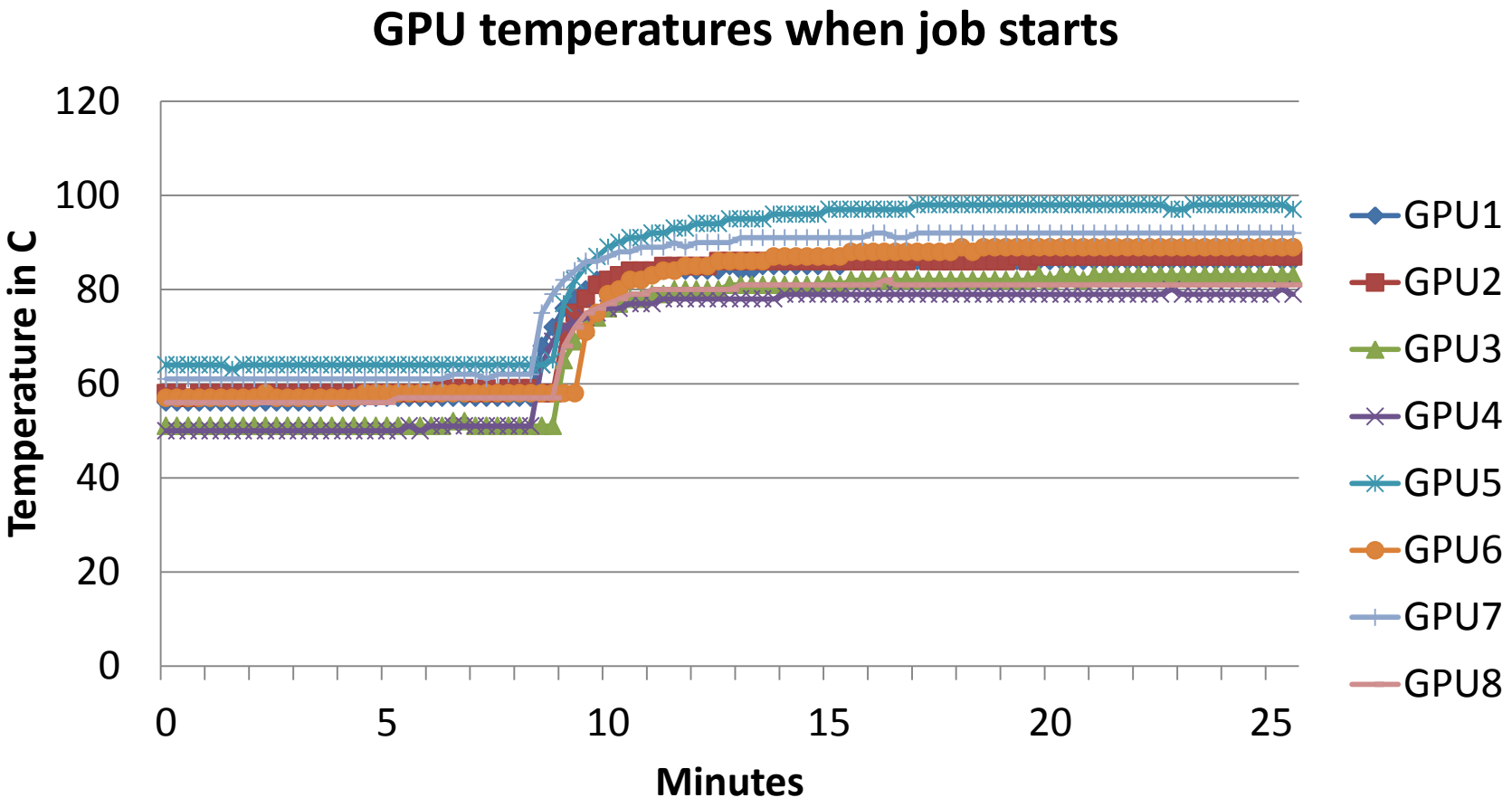
Production Issues

- Heat
- Task allocation
- Accuracy?
- Unplanned downtime
- Planned downtime

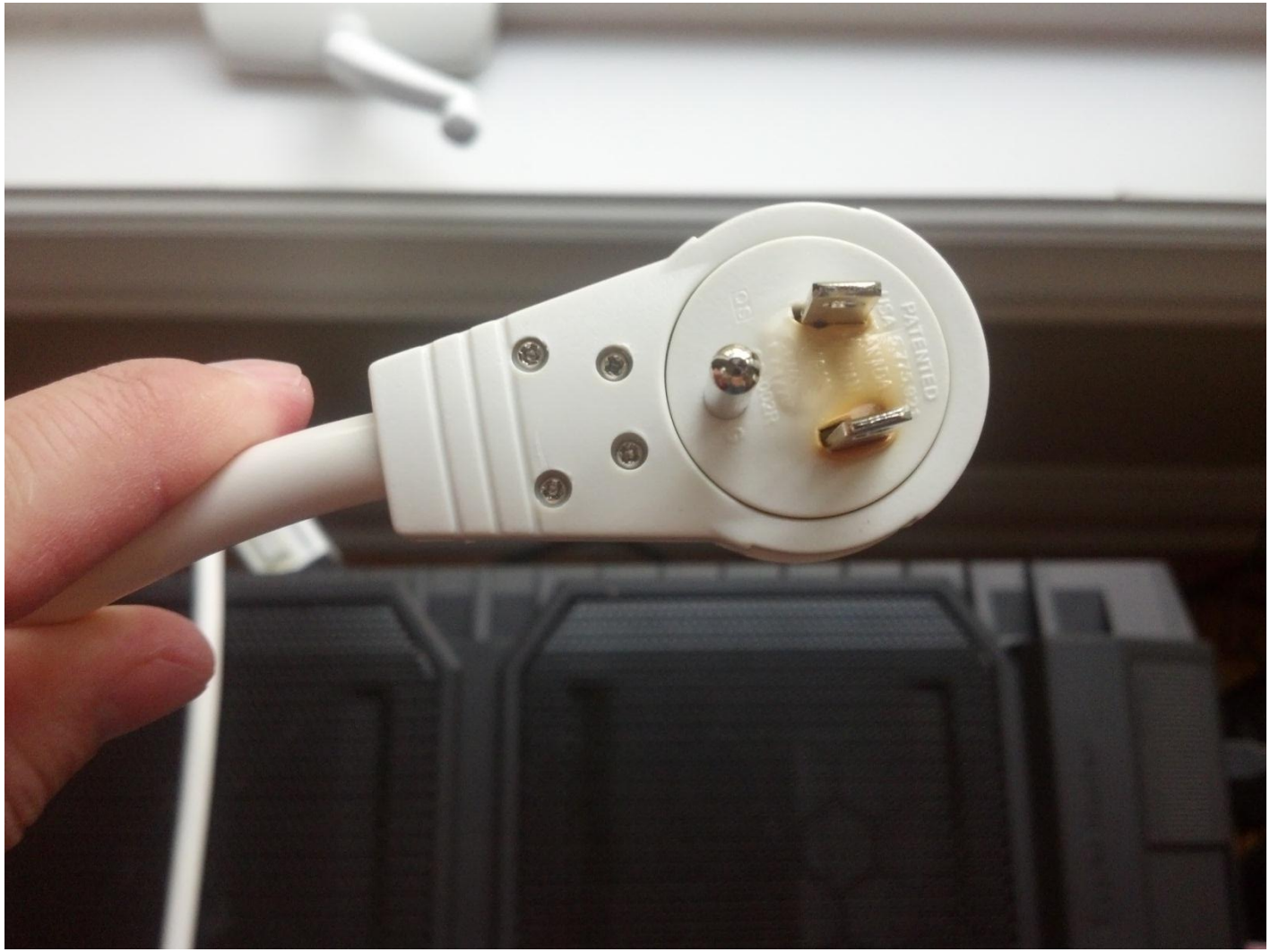
Homebuilt Box, 4xGTX 690



1100 Watt Space Heater



1800 Watt Power Strip



Accuracy

- How do I know the results are correct?
  - Shorter runs: check other results online
    - Bellard: up to 2.7T
    - Yahoo: up to 500T
  - Longer runs: run again, with offset

```
10^15-7: 1C23D488 353CB3F7 F0C9ACCF A8262A4B
10^15+1:          353CB3F7 F0C9ACCF A9AA215F 2556D630
```

Task Allocation

- Simple master/slave
- Each task ~1 minute long
  - Subrange of the main iteration loop
- Log per-task results

Per-task Checks

```
Iterations: 246,646,000,000,000..246,647,000,000,000
  digit: 1,999,999,999,999,999
    client: linux60821
    result: 0x1d57ca87472261f0d8735613bebd5721
  digit: 2,000,000,000,000,007
    client: linux60812
    result: 0x472261f0d8735613bebd15d037d6a42b

0x1d57ca87472261f0d8735613bebd5721
        0x472261f0d8735613bebd15d037d6a42b8
```

- Each task result can be double-checked
- On mismatch, just re-do two tasks

Hardware Errors?

- Over 50 days of GPU time
  - 4 GTX 690 (Kepler) cards: zero errors
  - 2 GTX 680 (Kepler) cards: zero errors
  - 24 GTX 570 (Fermi) cards:
    - 16 cards: zero errors
    - 4 cards: 1-4 errors
    - 4 cards: 7-10 errors
- All errors were irreproducible
- All consumer-level cards: non-ECC memory

Guerilla Supercomputing

- “Mostly” had permission
- Single-GPU machines
- Causes jittery display
- Solution? Hide.
  - Detect GUI login → idle the GPU

Useful Results

- Never need more than 38 digits of π
- Hardware stress-test
  - GTX 690: A+
  - GTX 570: B
  - Belkin power strip: F
- Reusable library
  - Multi-word integer library
  - Modular exponentiation

Next Steps

- More distant digits!
- Digits of π^2
- Other hardware / programming models
  - AMD, Intel MIC, BOINC
- Titan time?
- Looking for a good Ph.D. program…

Ed.karrels@gmail.com