# Parallelizing The Chinese Remainder Theorem

Professor Russ Miller
Bich Thi-Ngoc Vu
CSE 633 – Parallel Algorithms
Wednesday April 23, 2014

# CRT - Applications

- Cryptography (e.g. decryption in RSA)
- Computing
- Coding Theory
- Various others → useful for its ability to simply the problem.

# Chinese Remainder Theorem: Definition

The theorem can also be generalized as follows. Given a set of simultaneous congruences

$$x \equiv a_i \pmod{m_i}$$

for $i = 1, \ldots, r$ and for which the $m_i$ are pairwise relatively prime, the solution of the set of congruences is

$$x \equiv a_1 b_1 \frac{M}{m_1} + \ldots + a_r b_r \frac{M}{m_r} \pmod{M},$$

where

$$M = m_1 m_2 \cdots m_r$$

and the $b_i$ are determined from

$$b_i \frac{M}{m_i} \equiv 1 \pmod{m_i}.$$

(Courtesy of Mathworld)

# Chinese Remainder Theorem: Definition

The theorem can also be generalized a

1. Determine relative prime of $m_i$'s.

$$x \equiv a_i \pmod{m_i}$$

4. Calculate $x_i$'s and then X.

for $i = 1, \ldots, r$ and for wh… of the set of congruences is

$$x \equiv a_1\, b_1 \frac{M}{m_1} + \ldots + a_r\, b_r \frac{M}{m_r} \pmod{M},$$

where

2. Calculate M.

$$M = m_1\, m_2 \cdots m_r$$

and the $b_i$ are dete

3. Calculate $b_i$'s.

$$b_i \frac{M}{m_i} \equiv 1 \pmod{m_i}.$$

5. Verify X.

(Courtesy of Mathworld)

# Chinese Remainder Theorem: Example

- Given this set of linear congruences:
  - X = 1 mod 5
  - X = 2 mod 6
  - X = 3 mod 7

  Determine X.

X = 206 (mod 210)

| | | | |
|---|---|---|---|
| | **M = 5 * 6 * 7 = 210** | | |
| $a_1 = 1$ | $M_1 = 210/5 = 42$ | $b_1 = 3$ | 126 |
| $a_2 = 2$ | $M_2 = 210/6 = 35$ | $b_2 =5$ | 350 |
| $a_3 = 3$ | $M_3 = 210/7 = 30$ | $b_3 = 4$ | 360 |
| | | | 836 |

# Chinese Remainder Theorem: Example

1. Determine relative prime of $m_i$'s.

- Given this set of linear congruences:
  - X = 1 mod 5
  - X = 2 mod 6
  - X = 3 mod 7

Determine X.

5. Verify X with all congruences.

4. Calculate $x_i$'s and then X.

X = 206 (mod 210)

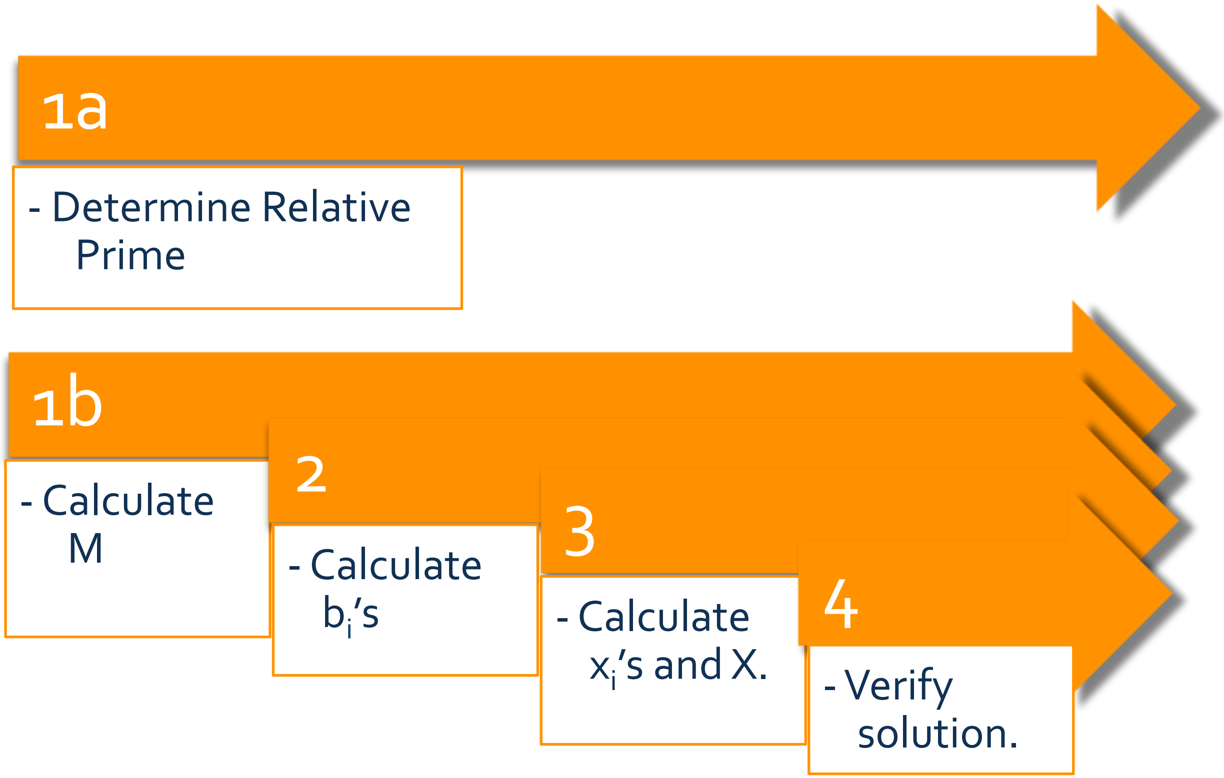
2. Calculate M.

**M = 5 * 6 * 7**
**= 210**

3. Calculate $b_i$'s.

| | | | |
|---|---|---|---|
| $a_1 = 1$ | $M_1 = 210/5 = 42$ | $b_1 = 3$ | 126 |
| $a_2 = 2$ | $M_2 = 210/6 = 35$ | $b_2 =5$ | 350 |
| $a_3 = 3$ | $M_3 = 210/7 = 30$ | $b_3 = 4$ | 360 |
| | | | 836 |

# Dependencies



# Implementation – Problem space

- Originally wanted to have a large number of congruences in one problem → values became very large very quick.
- Large number of problems instead.
  - Sets of 5 congruences per problem.
  - First 50 primes as $m_i$ values.
  - Range from 0-9 for $a_i$ values.
  - Recall: Equation is of the form

    $x = a_i \bmod m_i$

X of Problem 0 = 1523
X of Problem 1 = 352553
X of Problem 2 = 6.69896e+07
X of Problem 3 = 5.56846e+08
X of Problem 4 = 7.65351e+08
X of Problem 5 = 1.31566e+09
X of Problem 6 = 6.72314e+09
X of Problem 7 = 2.63586e+10
X of Problem 8 = 7.45597e+09
X of Problem 9 = 3.51302e+11

# Implementation Flow

1. Sequential
2. Explored multithreaded implementation. (Was curious)
3. Parallel for one complex problem.
   - Steps of determining relative prime, computing M, $b_i$'s, $x_i$'s, and X, and verifying solution were divided among processors.
4. Parallel for large number of problems.

# Parallel for One Complex Problem

- Each node gets approximately (number of congruences)/(number of nodes) to work with.
- F(noc, non, rank), where noc = no. of congruences and non = no. of nodes, at each of the steps to determine range of responsibility.
- Values were so large that solutions weren't represented properly.
  - -nan isn't very useful
- Wanted runtime in seconds.

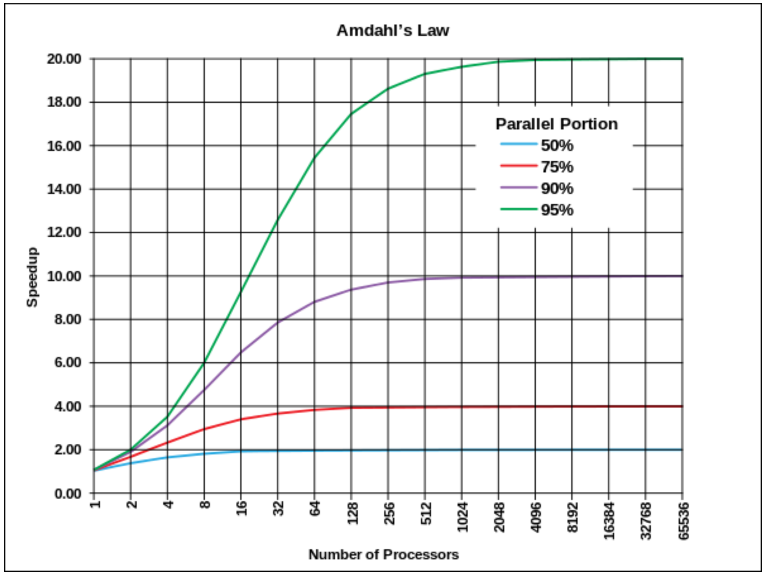
**Problem 0:** X = 1 mod 2
X = 2 mod 3
X = 3 mod 5
X = 4 mod 7
X = 5 mod 11
**Solution: X = 1523 mod 2310**
**Took: 0.021911 (MPI Time)**
**0.01 (C Time)**

# Parallel for Large Problem Set

- Each node gets approximately (number of problems)/(number of nodes) to work with.
- F(nop, non, rank) to determine range of responsibility.
- All nodes have an allocated problem set and need only worry about its domain of responsibility.
- For run time observations, less of a need to optimize within each problem.

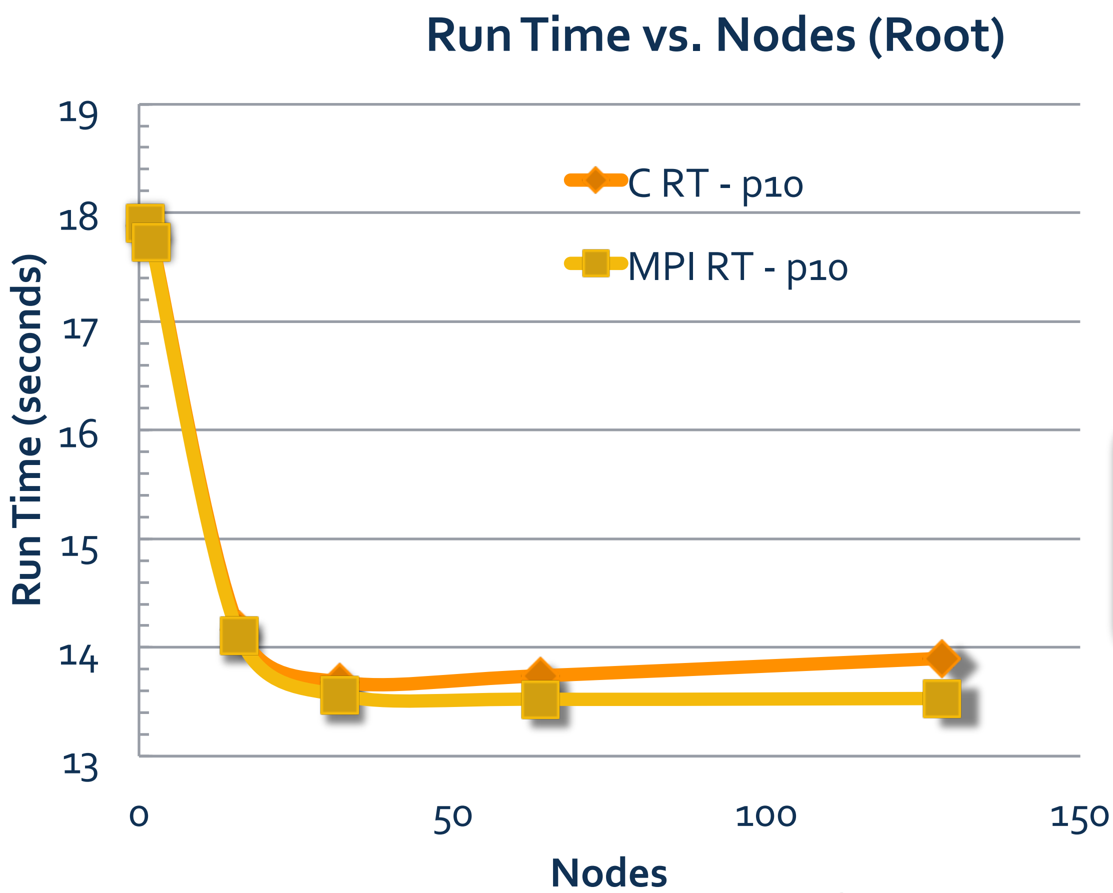
# Things to consider

- According to Amdahl's Law, minimum run time can not be better than the non-parallelizable part.
- Here, the non-parallelizable part has been set to be one problem (a set of 5 congruences).




(Image courtesy of Wikipedia)

# Running -> Only Root Node Needs All Solutions



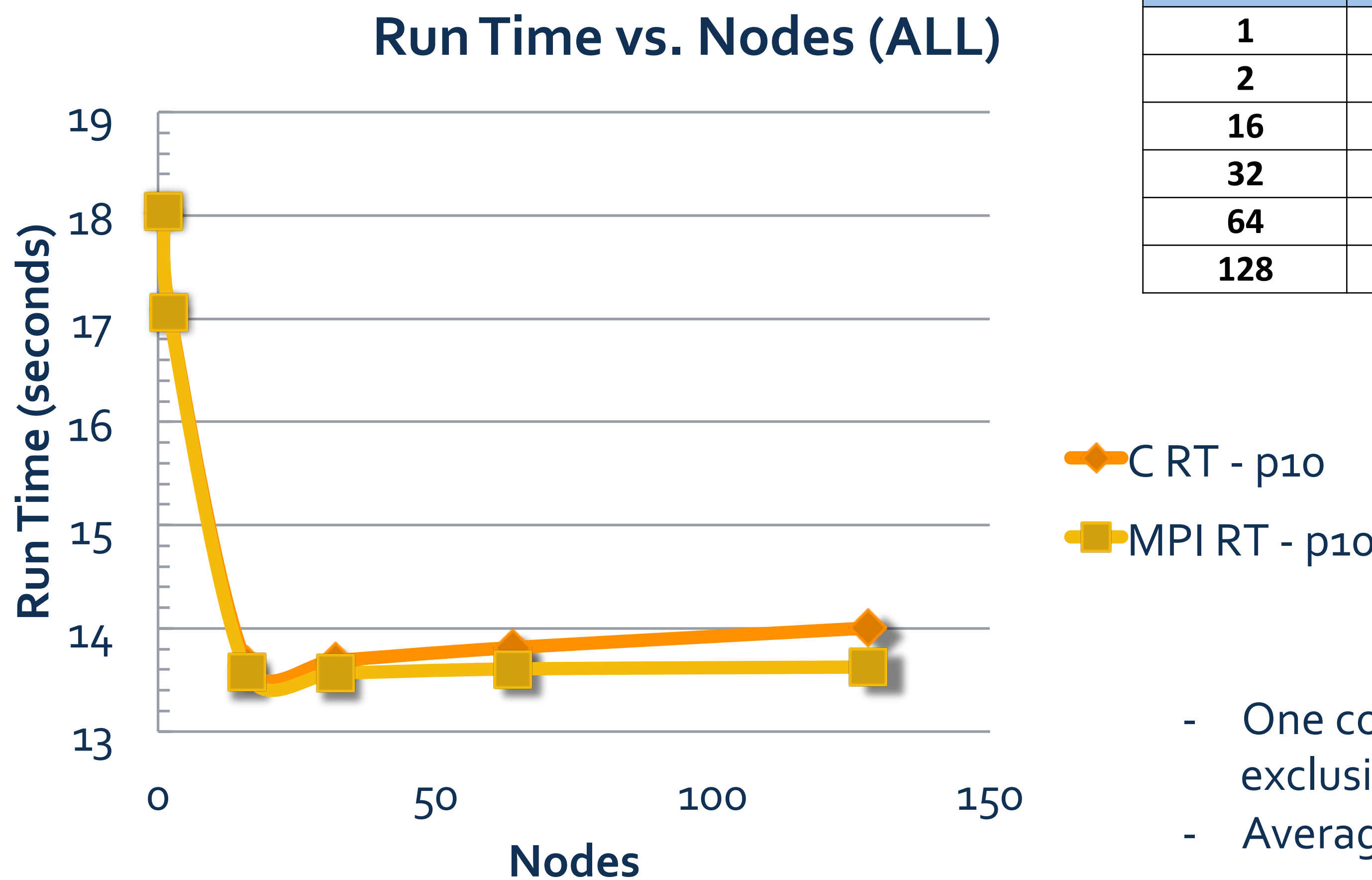

| Nodes | C RT - p10 | MPI RT - p10 |
|---|---|---|
| **1** | 17.880 | 17.901 |
| **2** | 17.753 | 17.727 |
| **16** | 14.163 | 14.105 |
| **32** | 13.683 | 13.560 |
| **64** | 13.740 | 13.524 |
| **128** | 13.900 | 13.531 |

What if ALL Nodes need to know ALL Solutions? More communication?

- One core / node with exclusive flag.
- Average of 3 runs.

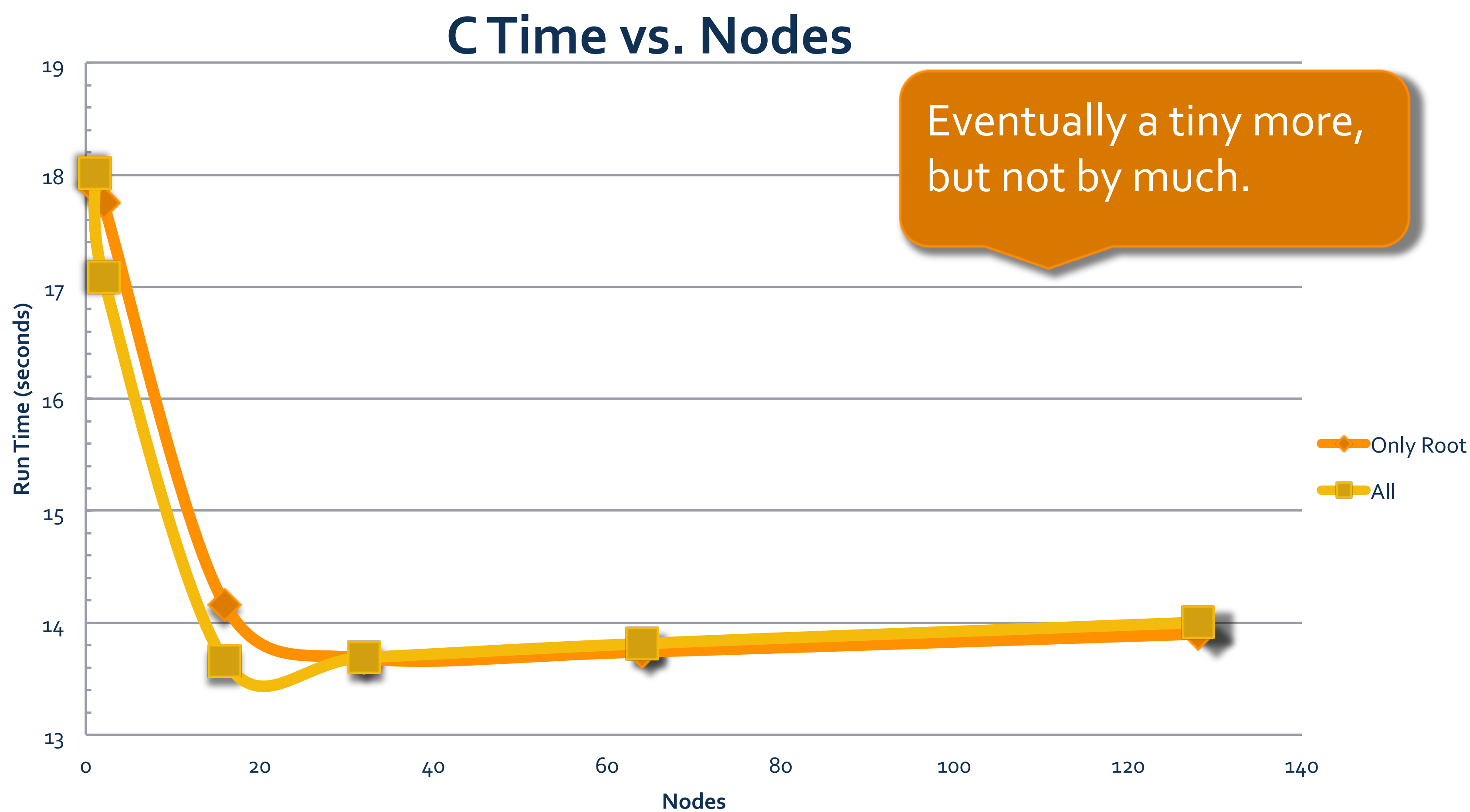
# Running -> All Nodes Need All Solutions

| Nodes | C RT - p10 | MPI RT - p10 |
|---|---|---|
| **1** | 18.020 | 18.037 |
| **2** | 17.090 | 17.062 |
| **16** | 13.657 | 13.576 |
| **32** | 13.690 | 13.567 |
| **64** | 13.813 | 13.607 |
| **128** | 14.003 | 13.625 |

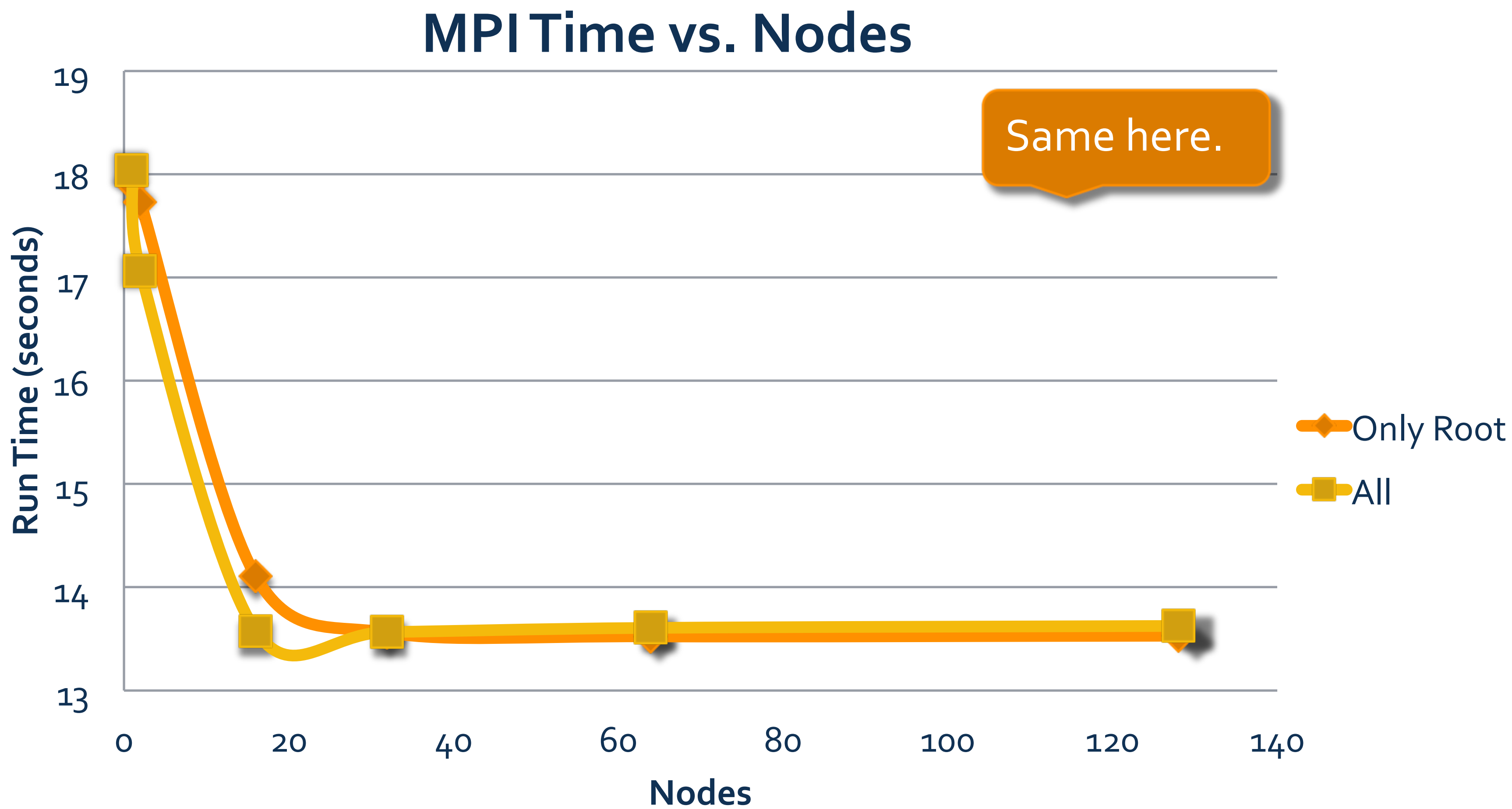



- One core / node with exclusive flag.
- Average of 3 runs.

# For better comparison – C Times



# For better comparison – MPI Times



# Lessons Learned

- Running programs with MPI in C.
- CRT is use to simplify more complicated problems so it has to be inherently fast.
- Since we are limited by the possible complexity (number of congruences per problem), considering a larger set (number of problems) for the problem space can do the trick .
- MPI_Allreduce (Max) is great for finding longest run time.

# References


- Weisstein, Eric W. "Chinese Remainder Theorem." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/ChineseRemainderTheorem.html
- Frey, Cathy. "The Chinese Remainder Theorem." Online video clip. *YouTube*. YouTube, 11 Jun. 2012. Web. 26 Feb. 2014.
- Liu, Jane. "Chinese Remainder Theorem: A look at its usage, proof, and applications." Web Resource. https://files.nyu.edu/jl860/public/crt.htm
- Olagunju, Amos O. "A Computational Exploration of the Chinese Remainder Theorem." From J. Appl. Math and Informatics, Vol. 26 (2008), No. 1-2, pp. 307-316.
- Amdahl's Law Graph: http://en.wikipedia.org/wiki/File:AmdahlsLaw.svg

# Thank you!

Questions ?