Pacific Northwest
NATIONAL LABORATORY
Proudly Operated by Battelle Since 1965

Towards Real-Time High-Performance Computing for Power Grid Analysis

PETER HUI, SATISH CHIKKAGOUDAR

Pacific Northwest National Laboratory

Richland, WA

Project Overview

Pacific Northwest NATIONAL LABORATORY
Proudly Operated by Battelle Since 1965

- Project under Pacific Northwest National Laboratory's "Future Power Grid Initiative"
- Goal: Study methods for achieving real-time computation using High-Performance Computing
- Using power-grid applications as a vehicle for this research
- Grid is evolving. Problem: data size!
  - PMUs, Smart meters: projected to generate massive amounts of streaming sensor data (O(TB/day)).
  - Data needs to be analyzed in real-time
- Target application: Dynamic State Estimation
  - Roughly:
    - Power grid states include rotor angle and speed at each generator
    - State not directly observable- measured quantities are functions of states
    - Therefore: we estimate state using a recursive dynamic state estimation
    - Matrix Multiplication plays a large part (thus current focus)
    - Real time constraint: Results need to keep up with PMU data rate (~30 ms)

Dynamic State Estimation

- Goal: to dynamically estimate the states of the power grid system by integrating the process model and data assimilating the measured quantities (i.e., PMU data) as they are available (i.e., in real-time)
  - Two mathematical models are needed: the physical/process and measurement models
- Caveat: physical and measurement models are not known exactly

Process model: $x_k = f_k(x_{k-1}) + w_{k-1}, \quad w_{k-1} \sim N(0, Q_{k-1})$

white noise added

Measurement model: $y_k = h_k(x_k) + v_k, \quad v_k \sim N(0, R_k)$

- To capture the effects of the noise, a statistical approach is taken, the Ensemble Kalman Filter
  - Core computational component of this approach are the matrix-matrix products

Ensemble Kalman Filter (Additive Noise)

Initialize, taking (N+1) samples

$$\hat{x}_{i,0|0} = \hat{x}_{0|0} + w_{i,0} \quad i = 0,1,\cdots,N$$

Prediction (process model)

$$\hat{x}_{i,k|k-1} = f_k(\hat{x}_{i,k-1|k-1}) + w_{i,k} \quad i = 0,1,\cdots,N$$

$$\bar{x}_{k|k-1} = \frac{1}{N}\sum_{i=1}^{N}\hat{x}_{i,k|k-1}$$

$$P^{x,h}_{k|k-1} = \frac{1}{N-1}\sum_{i=1}^{N}[\hat{x}_{i,k|k-1} - \bar{x}_{k|k-1}][h_k(\hat{x}_{i,k|k-1}) - \overline{h_k(\hat{x}_{k|k-1})}]^t$$

$$P^{h,h}_{k|k-1} = \frac{1}{N-1}\sum_{i=1}^{N}[h_k(\hat{x}_{i,k|k-1}) - \overline{h_k(\hat{x}_{k|k-1})}][h_k(\hat{x}_{i,k|k-1}) - \overline{h_k(\hat{x}_{k|k-1})}]^t$$

Correction (data assimilation)

$$K_k = P^{x,h}_{k|k-1}\left(P^{h,h}_{k|k-1} + R_k\right)^{-1}$$

$$\tilde{y}_{i,k} = z_k + v_{i,k} - h_k(\hat{x}_{i,k|k-1}) \quad i = 0,1,\cdots,N$$

$$\hat{x}_{i,k|k} = \hat{x}_{i,k|k-1} + K_k\tilde{y}_{i,k} \quad i = 0,1,\cdots,N$$

Matrix-matrix products

Finalize

$$\bar{x}_{k|k} = \frac{1}{N}\sum_{i=1}^{N}\hat{x}_{i,k|k} \qquad P_{k|k} = \frac{1}{N-1}\sum_{i=1}^{N}(\hat{x}_{i,k|k} - \bar{x}_{k|k})(\hat{x}_{i,k|k} - \bar{x}_{k|k})^t$$

Real-Time HPC - Challenges

- Usual real-time challenges
  - Process Scheduling
  - Memory Management
  - Resource Allocation
  - …
- HPC-specific challenges
  - Internode communication
  - Timing interactions between concurrent processes

Real-Time HPC - Challenges

- Usual real-time challenges
  - Process Scheduling
  - Memory Management
  - Resource Allocation
  - …
- HPC-specific challenges
  - Internode communication
  - Timing interactions between concurrent processes

RTOS on each node

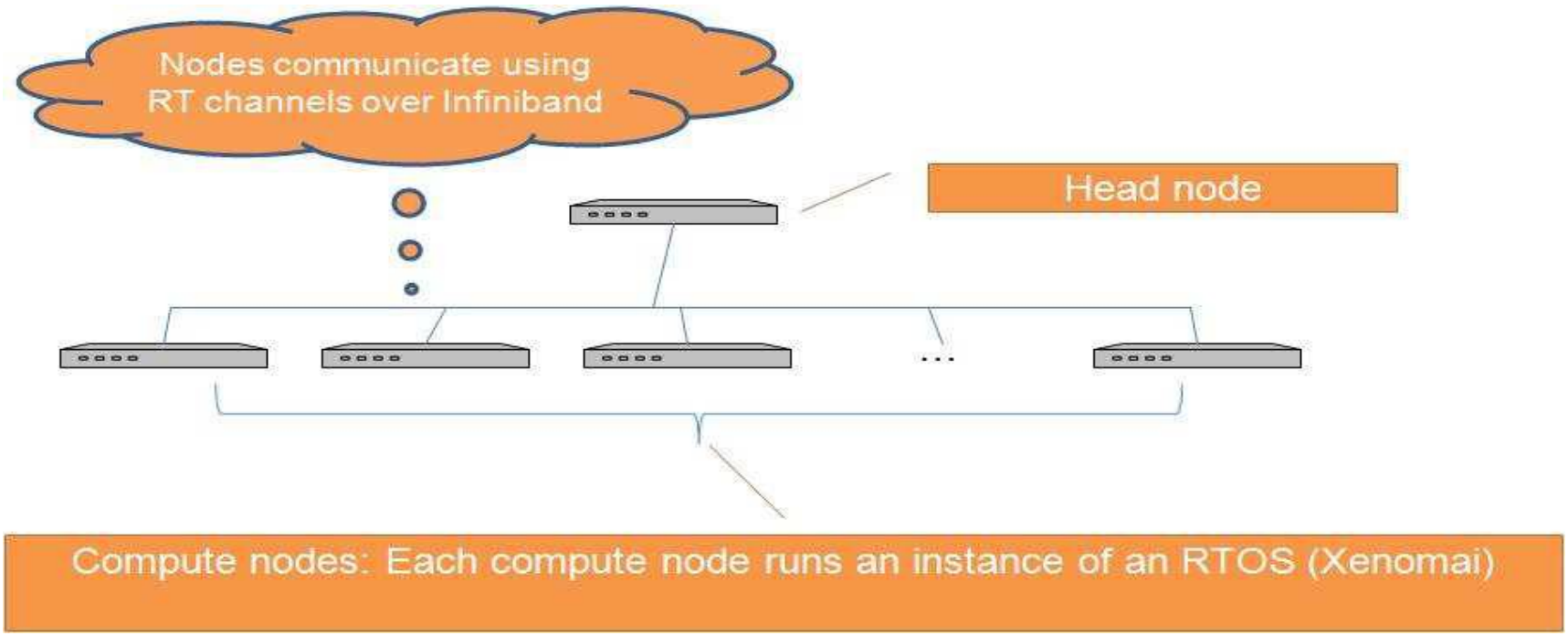
Infiniband drivers recompiled into RT space

Verification via formal model*

* Peter Hui, Satish Chikkagoudar, "A Formal Model for Real-Time Parallel Computation". In Proceedings of First International Workshop on Formal Techniques for Safety-Critical Systems (FTSCS'12), Kyoto, Japan. 2012.

Infrastructure

- Start with a standard cluster
- Modifications
  - Compute nodes run RTOS (i.e. Xenomai)
  - Need real time data transport over Infiniband (IB)

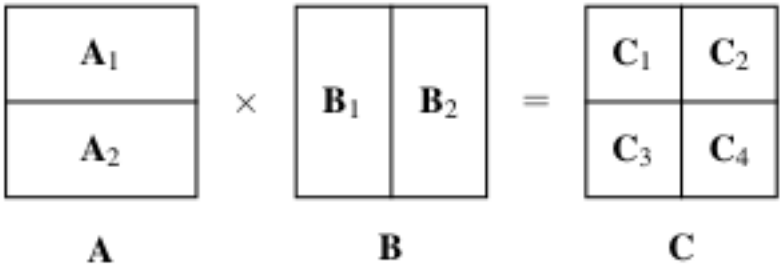


Progress

- Dynamic State Estimation application (Lee)
- Focus on one kernel: extracted Matrix Multiplication from application
  - Simple implementation
  - Compiled using Xenomai real-time framework
  - Control flow, concurrent process interaction modeled
  - Timing measurements taken, fit into model
  - Analysis gives provable upper bound on runtime
  - Result: 128x128 matrix multiplication in < 9 ms
    - ** NOTE: size, time is not a surprising result
    - Main results of interest:
      - Provable upper bound
      - Analysis, development process, thought process behind RT cluster computing

Kernel Focus: Matrix Multiplication

- Algorithm



- Example: Four processor cluster
  - P1: Computes C1 = A1 x B1
  - P2: Computes C2 = A1 x B2
  - P3: Computes C3 = A2 x B1
  - P4: Computes C4 = A2 x B2

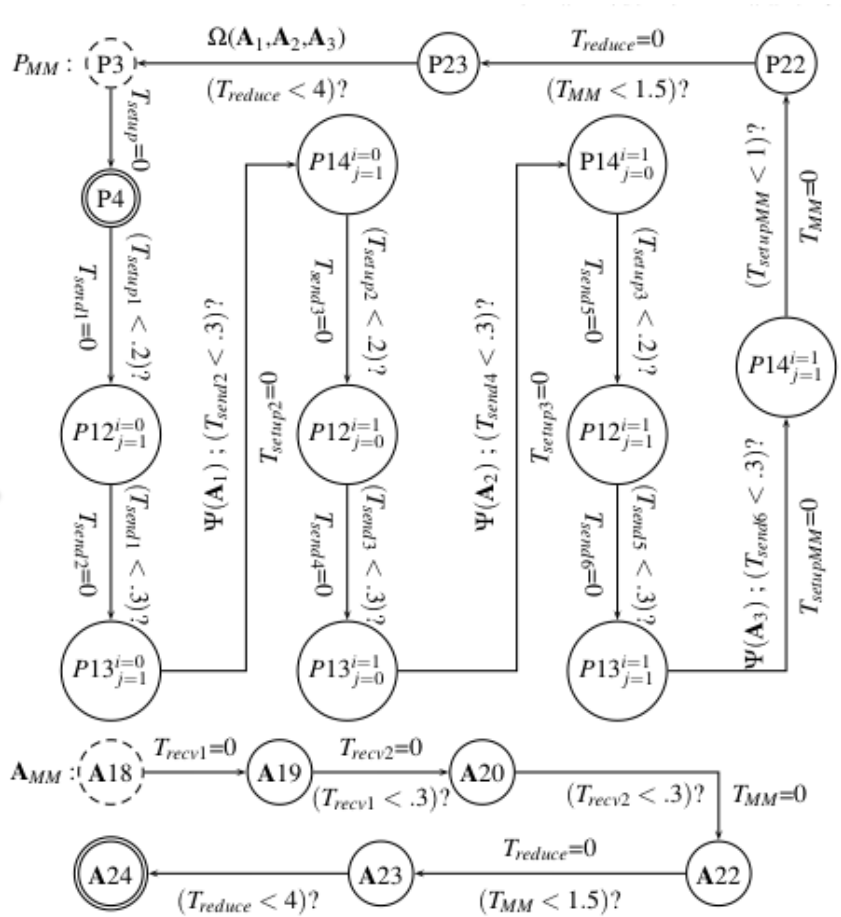
Matrix Multiplication – Timing Model (I)

- At a high level: Derive annotated control flow graph from algorithm

Algorithm 1 MatrixMultiply: Compute $\mathbf{C} = \mathbf{A} \times \mathbf{B}$

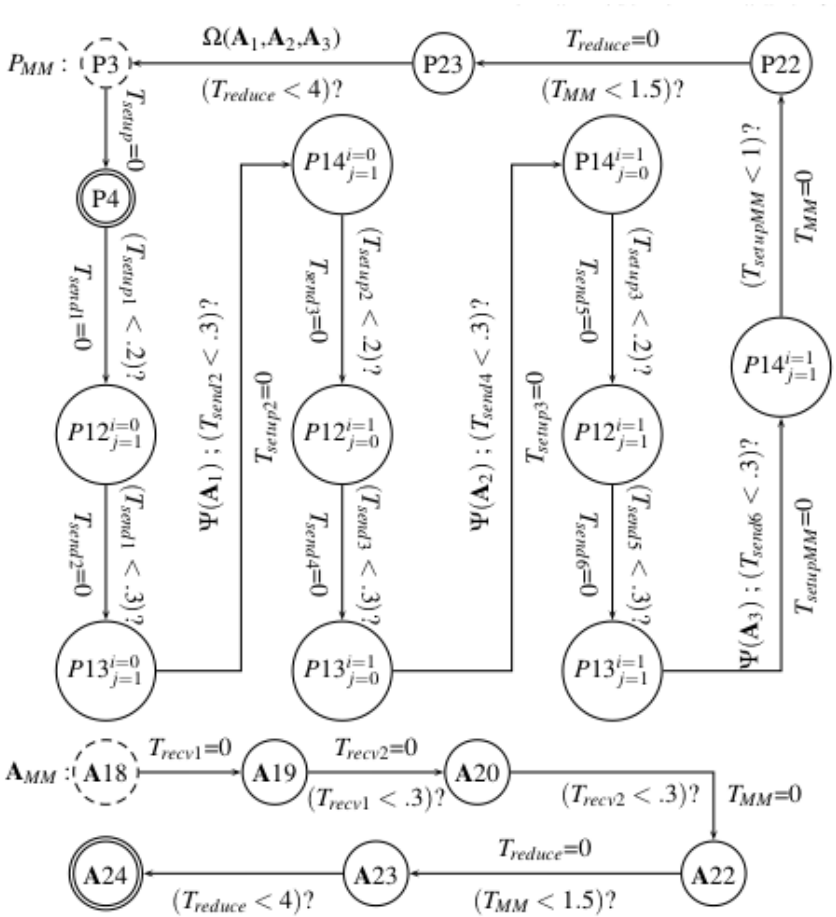
p : Number of processors
N : Rank of matrices

$q \leftarrow \sqrt{p}$
while true do
  $dest \leftarrow 1$
  if self $== 0$ **then** {Master process}
    for $i = 0$ to $q-1$ **do**
      for $j = 0$ to $q-1$ **do**
        $w \leftarrow i\frac{N}{q}, x \leftarrow (i+1)\frac{N}{q}$
        $y \leftarrow j\frac{N}{q}, z \leftarrow (j+1)\frac{N}{q}$
        $\tilde{X} \leftarrow \mathbf{A}[w:x][0:N]$
        $\tilde{Y} \leftarrow \mathbf{B}[0:N][y:z]$
        if $i \neq 0$ **and** $j \neq 0$ **then** {Master already has these chunks}
          send$(\tilde{X}, dest)$
          send$(\tilde{Y}, dest)$
          $dest \leftarrow dest + 1$
        end if
      end for
    end for
  else {Child processes}
    $\tilde{X} \leftarrow$ **recv**(0)
    $\tilde{Y} \leftarrow$ **recv**(0)
  end if
  $\tilde{Z} \leftarrow$ **locMM**$(\tilde{X}, \tilde{Y})$
  reduce$(\tilde{Z}, \mathbf{C})$
end while



Matrix Multiplication – Timing Model (II)

- Annotated CFG allows us to reason about timings and interactions between processes
- Top CFG: Parent process
- Bottom CFG: Child process
- States: Correspond to key points in algorithm
- Ψ(A) : Fork (spawn) child process along transition
- Ω(A) : Join with previously spawned child process
- Timers constrain transitions based on delays
- Main result: Provable upper delay of 9 ms per cycle (under specific conditions)



Next Steps

- Integrate this kernel into main Dynamic State Estimation application
- Perform similar analysis on main application
- How to automate this control flow analysis?
  - Possible approach: Compiler driven?

QUESTIONS?