Preliminary Study toward Intelligent Run-time Resource Management Techniques for Large Tiled Multi-Core Architectures

Dong-In Kang, Jinwoo Suh,
Janice O. McMahon, and Stephen P. Crago

University of Southern California
Information Sciences Institute

September 18, 2007

This effort was sponsored by Defense Advanced Research Projects Agency (DARPA) through the Dept. of Interior National Business Center (NBC), under grant number NBCH1050022. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsement, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), Dept. of Interior, NBC, or the U.S. Government.

Outline

- Introduction
- Simulation Model
- Applications and Results
  - □ Resource Fragmentation
  - □ Mapping Performance
- Problem Space and Future Work
- Conclusion

Approved for public release, distribution unlimited

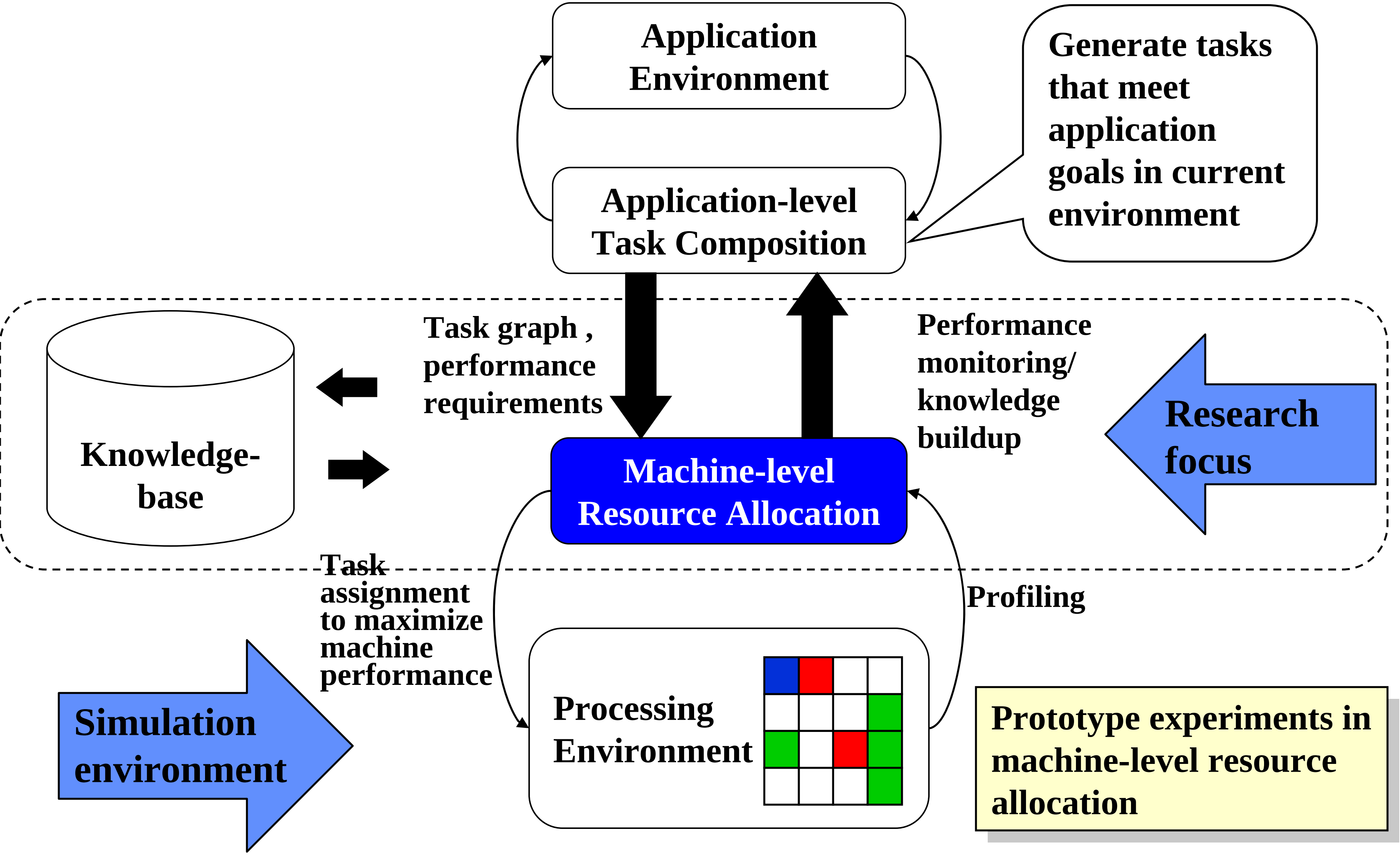
Motivation

- Rising processor density, multi-core architecture
  - MIT Raw 4x4
  - 8-core 64-thread SUN Niagara 2 processor
  - Tilera TILE64 processor
  - Intel 80-core Teraflops prototype processor
  - In the future: 32x32 or larger
- Dynamic applications
  - With large-scale architectures, multiple applications will share tile array; need to arbitrate resources between applications
  - Requirements are unpredictable and changing in response to environment
  - Need for dynamic mapping, allocation/deallocation, resource management
- Need for more intelligent run-time systems
  - Better resource allocation for efficiency
  - New system capabilities to alleviate programming burden
  - Dynamic reactivity for changing scenarios
  - Complex optimization space

Research Goal: Intelligent Run-Time Resource Allocation

- Application of knowledge-based and learning techniques to the problem of dynamic resource allocation in a tiled architecture
  - □ Planning and optimization of resource usage given application constraints
  - □ Knowledge derived from dynamic performance profiling, modeling, and prediction
  - □ Run-time learning of resource allocation trade-offs
- Proof-of-concept demonstration: simulation of application workload to compare performance of manual and automatic resource allocations
  - □ Define representative application
  - □ Develop scalable simulation
  - □ Define resource mapping problem and apply intelligent technique
- Current work: definition of problem, construction of simulator, and preliminary study to quantify performance effects

System Framework



Application Environment

Application-level Task Composition

Generate tasks that meet application goals in current environment

Task graph , performance requirements

Performance monitoring/ knowledge buildup

Knowledge-base

Machine-level Resource Allocation

Research focus

Task assignment to maximize machine performance

Profiling

Simulation environment

Processing Environment

Prototype experiments in machine-level resource allocation

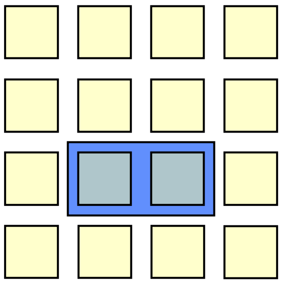
Approved for public release, distribution unlimited

Outline

- Introduction
- Simulation Model
- Applications and Results
  - Resource Fragmentation
  - Mapping Performance
- Problem Space and Future Work
- Conclusion

Approved for public release, distribution unlimited

ISI Simulator for Multi-Core Architecture

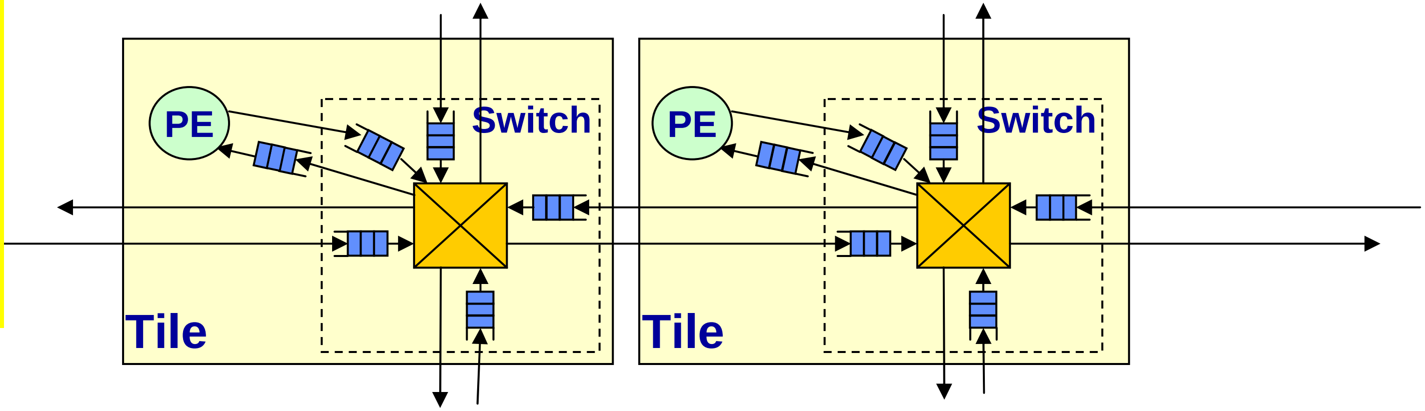


Profile info:
- **Per link:**
  - **Data amount**
  - **Blocking time**
- **Per packet:**
  - **Latency**
  - **Blocking time**
- **Per task:**
  - **Event log**
  - **Blocking time**
- **Per CPU:**
  - **Utilization**

Event-driven simulator
- In SystemC
- Based on ARMn multiprocessor simulator from Princeton Univ.

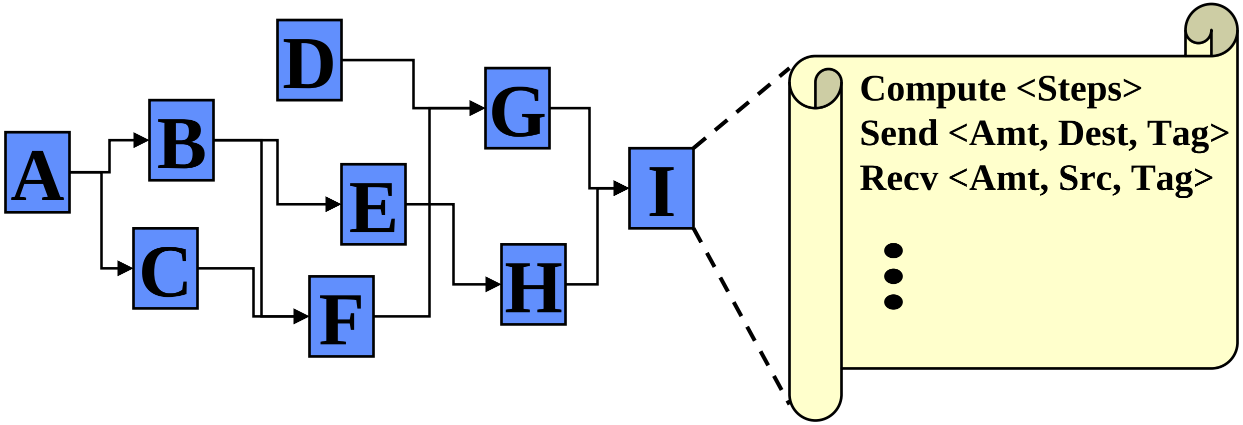
Generic processor module
- Proportional share resource scheduler
- Memory latency is implicit in computation time
- Parameterized specification: CPU speed, CPU-network interface speed, etc.

Network
- One network, 2-D mesh
- 5x5 crossbar switch
- Message passing model
- Store and forward routing
- Parameterized specification: speed, buffer size, packet overhead, etc.



Simulation Approach

Application Task Graph



Compute <Steps>
Send <Amt, Dest, Tag>
Recv <Amt, Src, Tag>
⋮

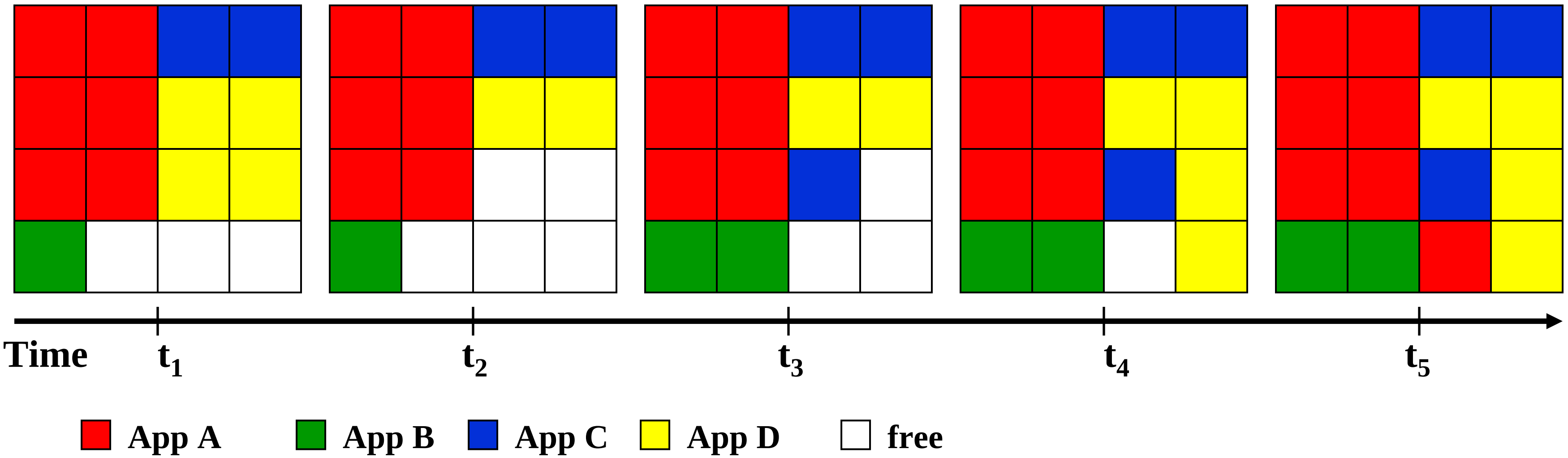
Each task has a script that defines computation steps, message sends and receives

Tile Allocation: 2D Mesh

A	A	B	B
C	D	D	E
F	F	F	G
G	H	I	I

Simulation uses data flow information and network characteristics (topology, latency, overhead, etc.) to produce performance results for a particular task mapping.

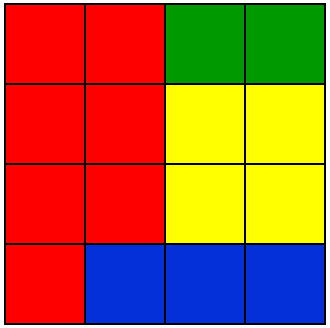
Dynamic Behavior



Two performance effects over time:

- Resource fragmentation: incremental resource allocation decreases locality
- Dynamic application requirements and resource availability

Optimized Mapping

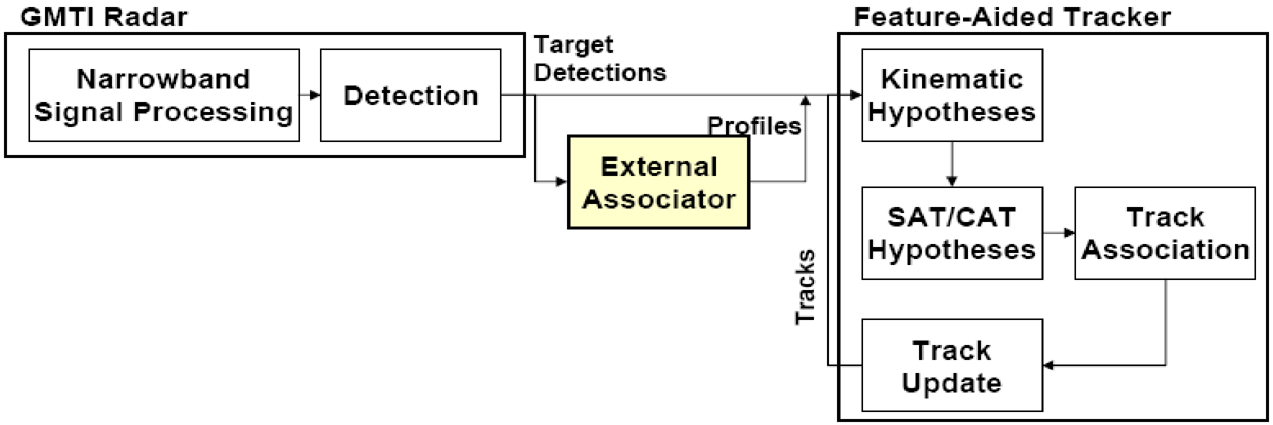


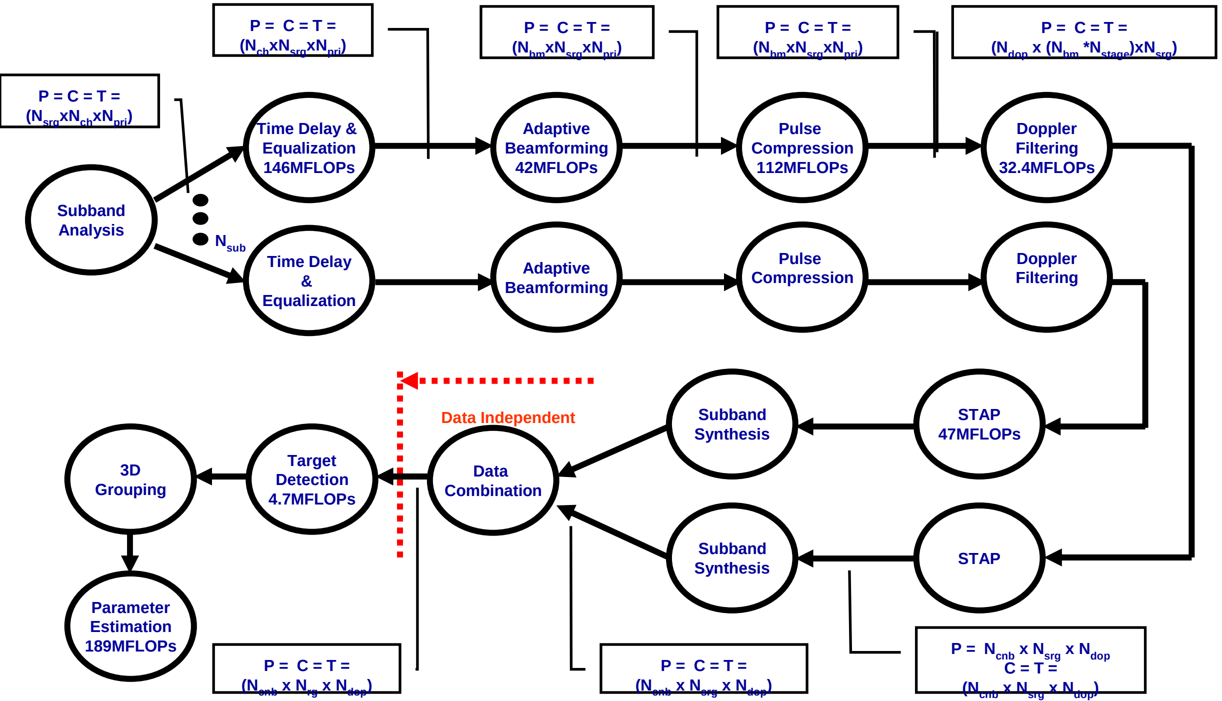
Outline

- Introduction
- Simulation Model
- Applications and Results
  - □ Resource Fragmentation
  - □ Mapping Performance
- Problem Space and Future Work
- Conclusion

Approved for public release, distribution unlimited

Example Application: Radar Resource Management using IRT



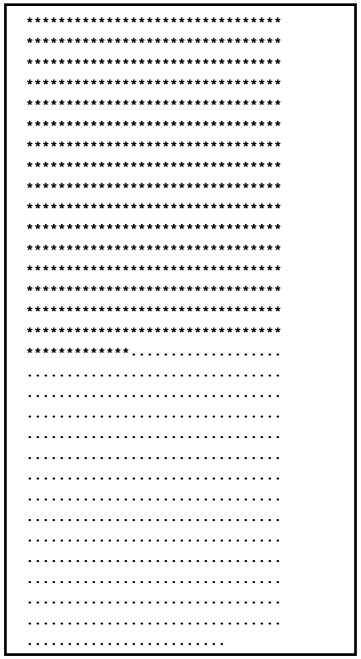

P = C = T = $(N_{srg}xN_{ch}xN_{pri})$
P = C = T = $(N_{ch}xN_{srg}xN_{pri})$
P = C = T = $(N_{bm}xN_{srg}xN_{pri})$
P = C = T = $(N_{bm}xN_{srg}xN_{pri})$
P = C = T = $(N_{dop}$ x $(N_{bm}$ $*N_{stage})xN_{srg})$
Subband Analysis
N_{sub}
Time Delay & Equalization 146MFLOPs
Adaptive Beamforming 42MFLOPs
Pulse Compression 112MFLOPs
Doppler Filtering 32.4MFLOPs
Time Delay & Equalization
Adaptive Beamforming
Pulse Compression
Doppler Filtering
Data Independent
Subband Synthesis
STAP 47MFLOPs
3D Grouping
Target Detection 4.7MFLOPs
Data Combination
Subband Synthesis
STAP
Parameter Estimation 189MFLOPs
P = C = T = $(N_{cnb}$ x N_{rg} x $N_{dop})$
P = C = T = $(N_{cnb}$ x N_{srg} x $N_{dop})$
P = N_{cnb} x N_{srg} x N_{dop} C = T = $(N_{cnb}$ x N_{srg} x $N_{dop})$

- Drive IRT spreadsheets and morphing scenarios with parameters from a notional radar resource manager
- GMTI Task Graph input to simulator
- FAT requirements modeled with statistically varying processor loads, message traffic

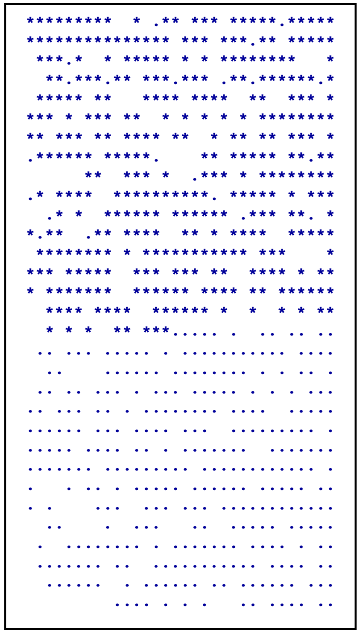
Resource Fragmentation

Fragmentation happens quickly!

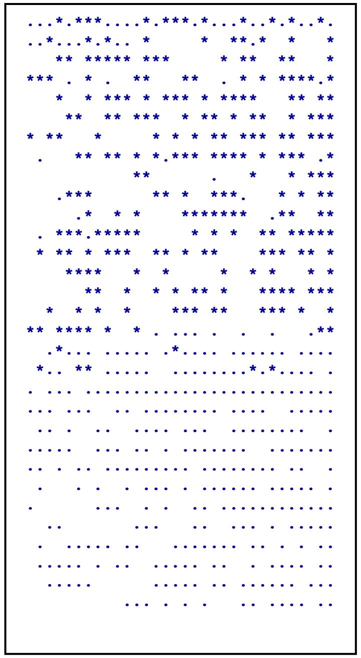
0: GMTI[525], FAT[460], FREE[39]
* GMTI asks to have [525] tiles
  Allocate GMTI [525] more nodes
* FAT asks to have [460] tiles
  Allocate FAT [460] more nodes



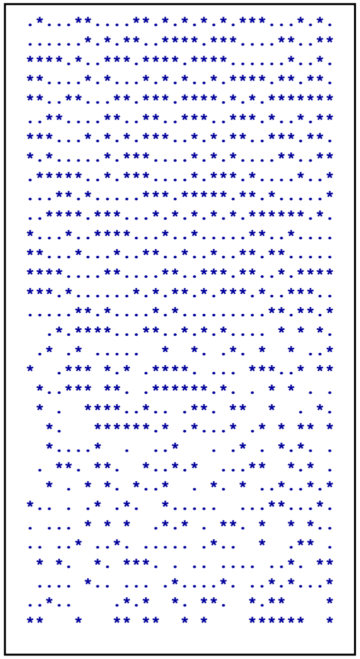
8: GMTI[367], FAT[361], FREE[296]
* GMTI asks to have [367] tiles
  Free GMTI [123] nodes



16: GMTI[251], FAT[355], FREE[418]'
* FAT asks to have [355] tiles
  Allocate FAT [7] more tiles



25: GMTI[419], FAT[446], FREE[159]
* GMTI asks to have [419] tiles
  Free GMTI [400] tiles
* FAT asks to have [446] tiles
  Allocate FAT [241] more tiles



‘*’: GMTI, ‘.’: FAT, ‘ ‘: FREE

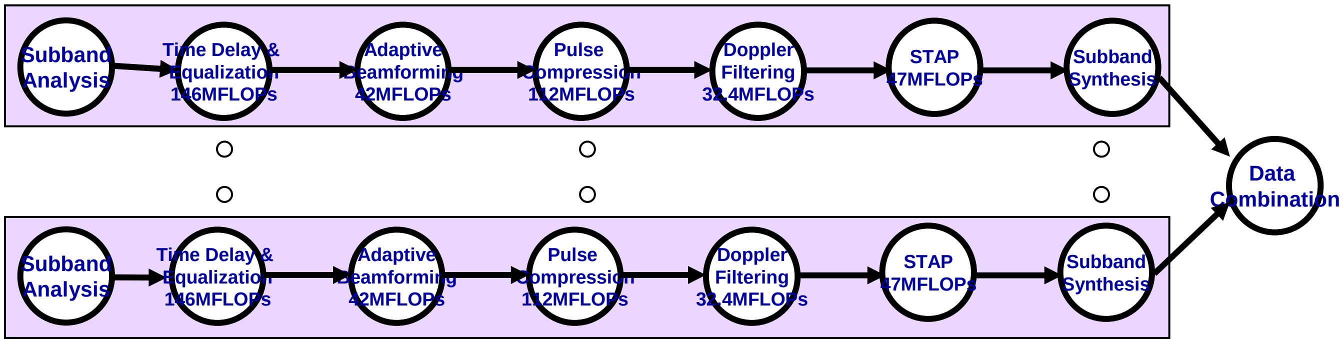
Assumptions:
- GMTI: mode change
- FAT: constrained in computation requirement changes in ±10%
- Linear tile assignment
- GMTI, FAT can change independently
- GMTI has higher priority than FAT

Outline

- Introduction
- Simulation Model
- Applications and Results
    - □ Resource Fragmentation
    - □ Mapping Performance
- Problem Space and Future Work
- Conclusion

Example: GMTI Application Mapping

Task Graph (GMTI)



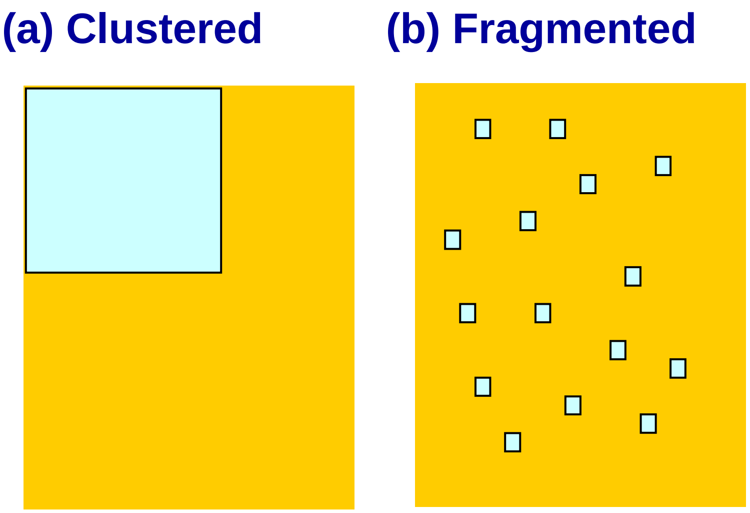
Subband Analysis → Time Delay & Equalization 146MFLOPs → Adaptive Beamforming 42MFLOPs → Pulse Compression 112MFLOPs → Doppler Filtering 32.4MFLOPs → STAP 47MFLOPs → Subband Synthesis → Data Combination

Subband Analysis → Time Delay & Equalization 146MFLOPs → Adaptive Beamforming 42MFLOPs → Pulse Compression 112MFLOPs → Doppler Filtering 32.4MFLOPs → STAP 47MFLOPs → Subband Synthesis → Data Combination

Interference from other application is modeled by random traffic generator

Parameter	Values	
GMTI resource requirements	128 tiles	868 tiles
Free tiles	Fragmented	Clustered
Mapping technique	Random	Heuristic
Task frequency	1 KHz	2 KHz

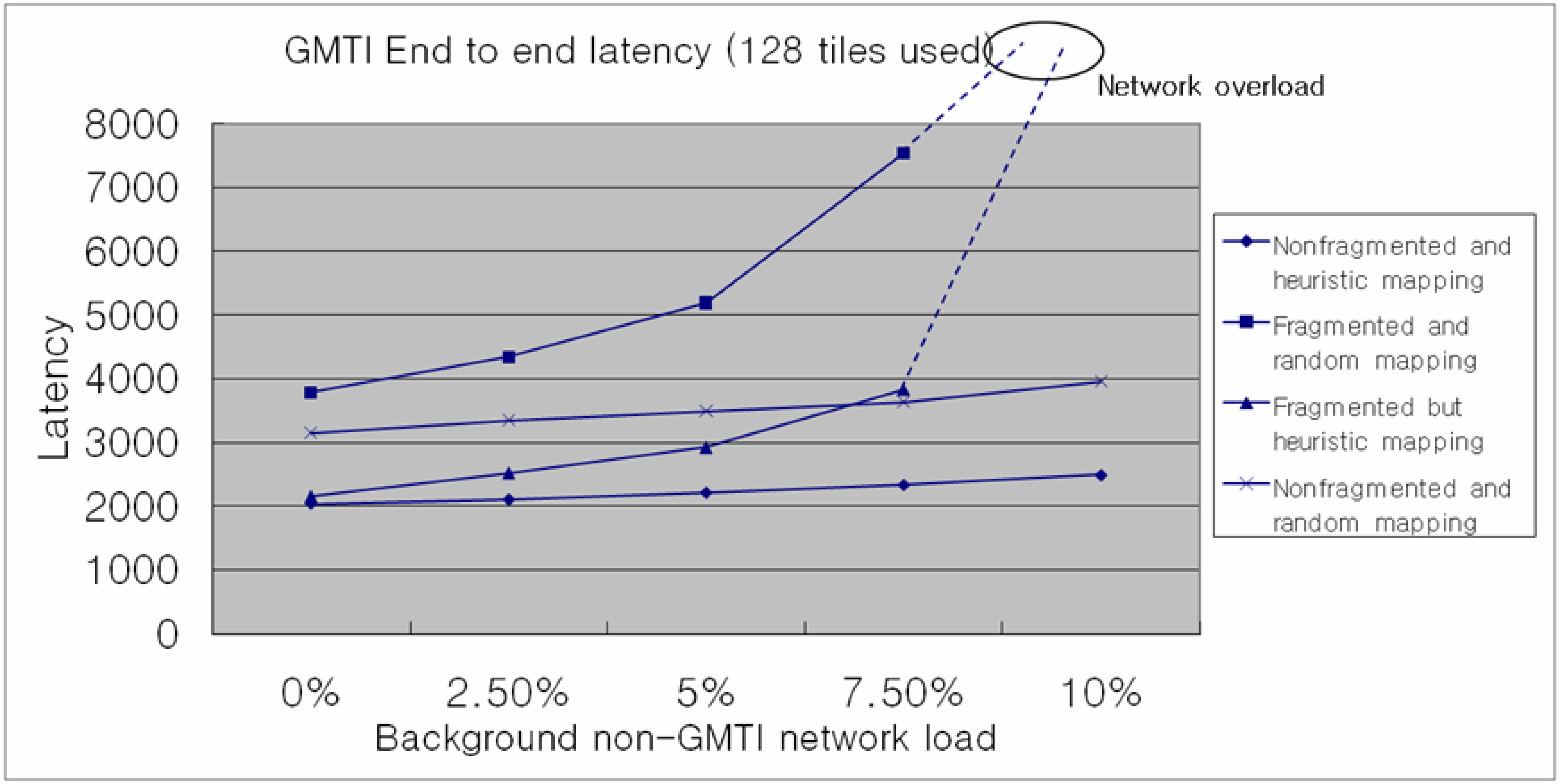
*** Total 16 cases**



(a) Clustered

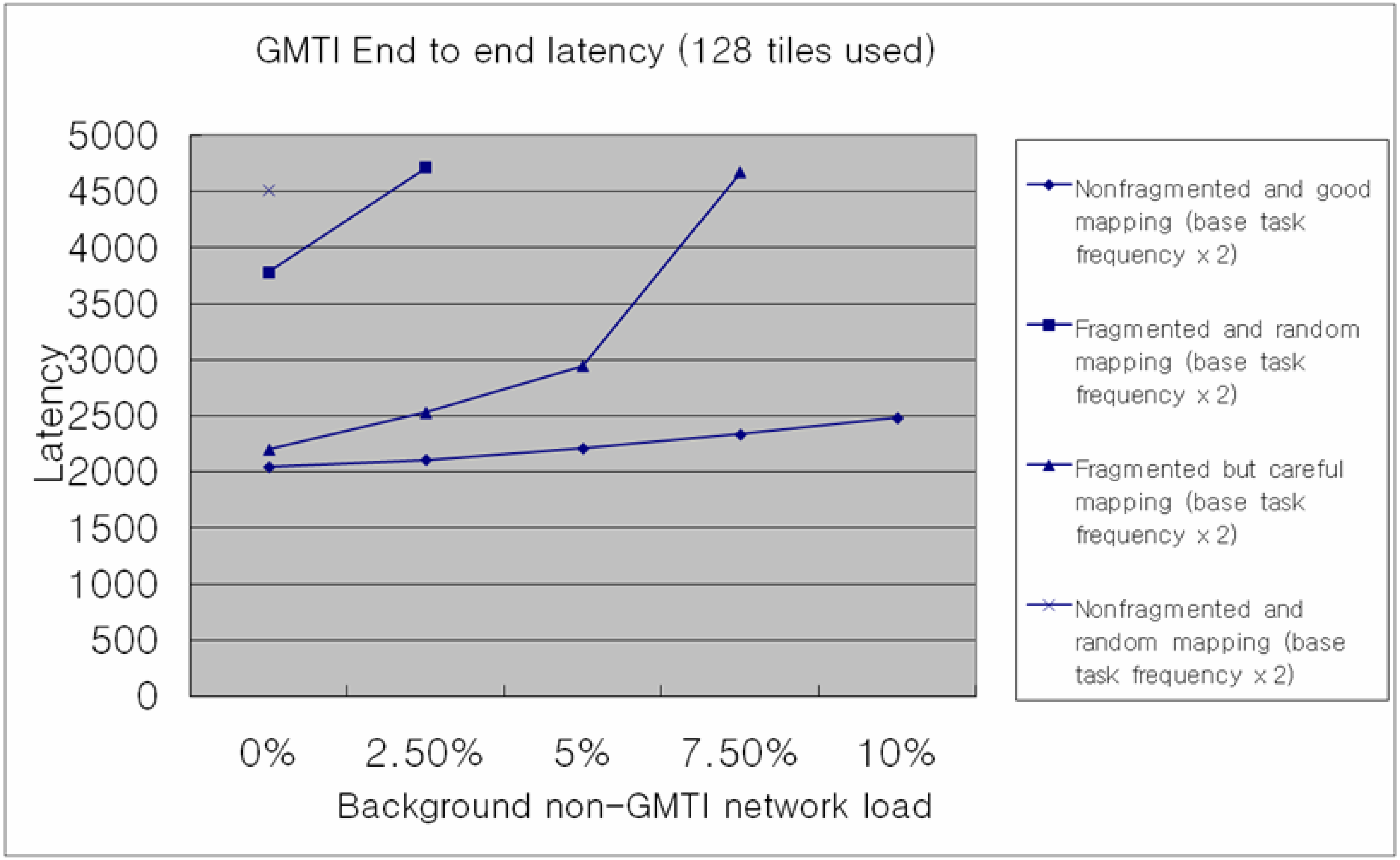
(b) Fragmented

128-Tile 1 KHz GMTI Performance



For high network loads, mappings on fragmented resources cause network load imbalance and longer latencies. In some cases, traffic causes network overload.

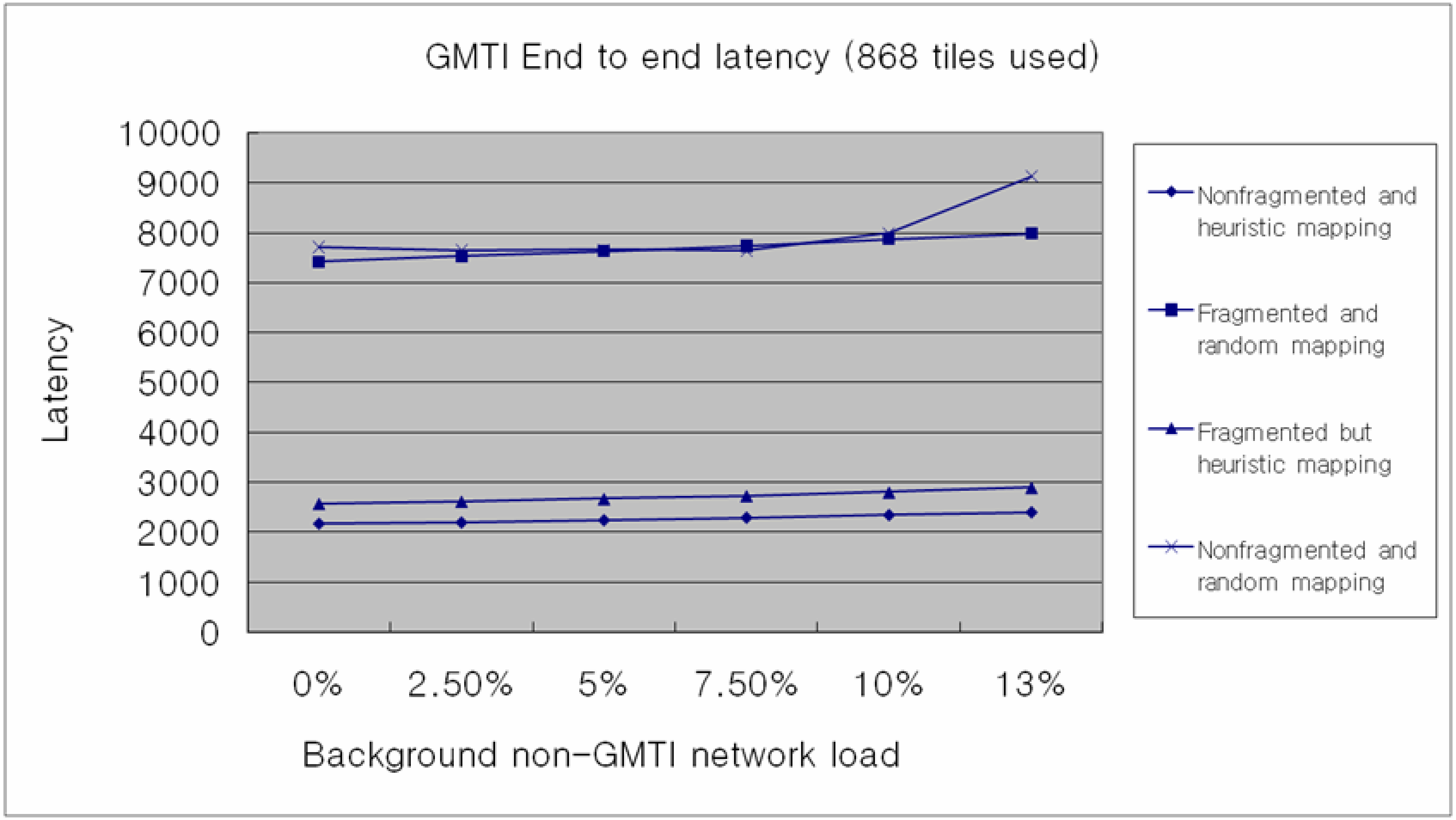
128-Tile 2 KHz GMTI Performance



With higher task frequencies (performance), latency is sensitive to both resource allocation method and fragmentation of the resources.

Approved for public release, distribution unlimited

868-Tile 1 KHz GMTI Performance



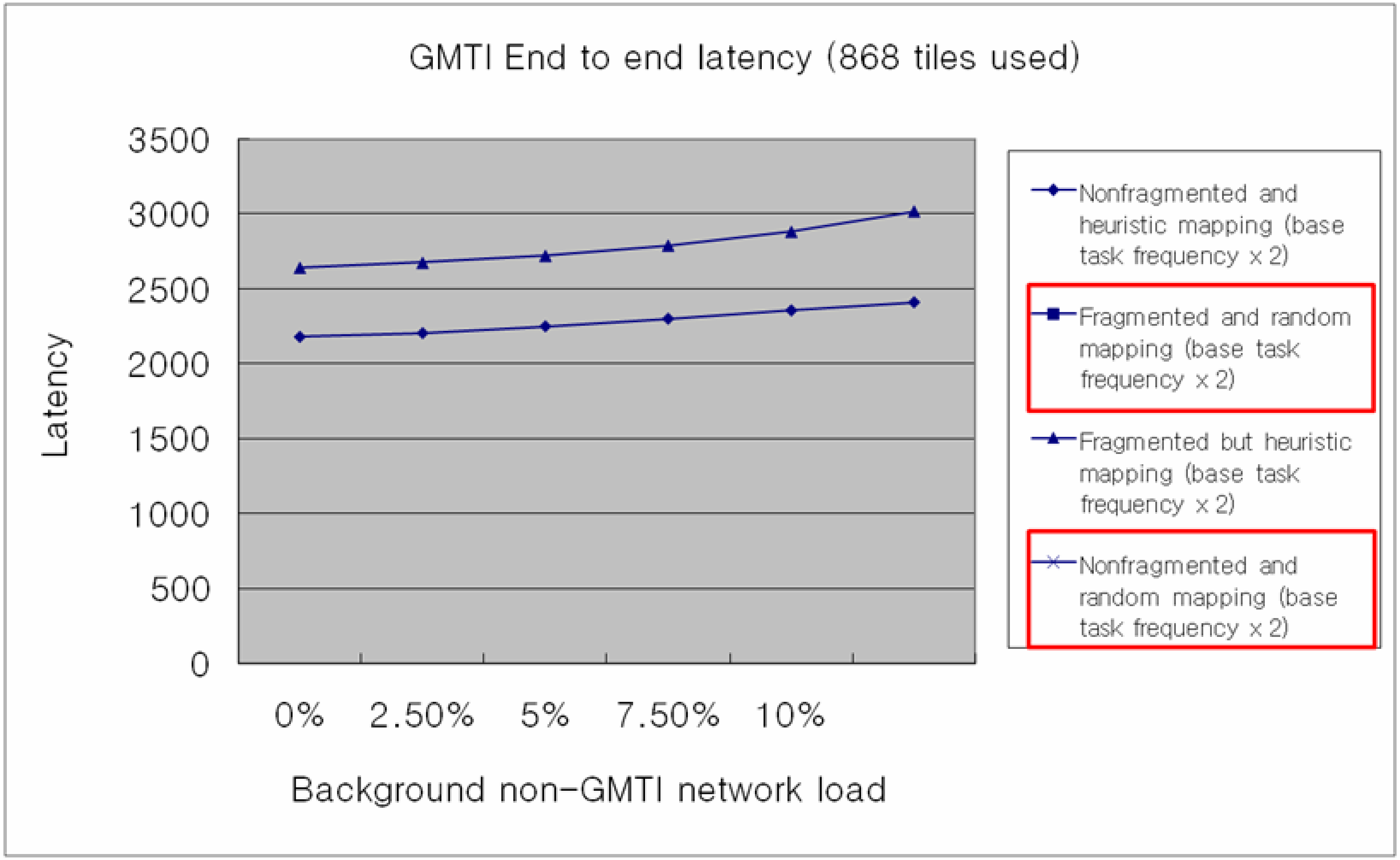
With higher GMTI allocation, latency is:

- less sensitive to background network load

- more sensitive to task mapping algorithm than fragmentation of the resources

Approved for public release, distribution unlimited

868-Tile 2 KHz GMTI Performance



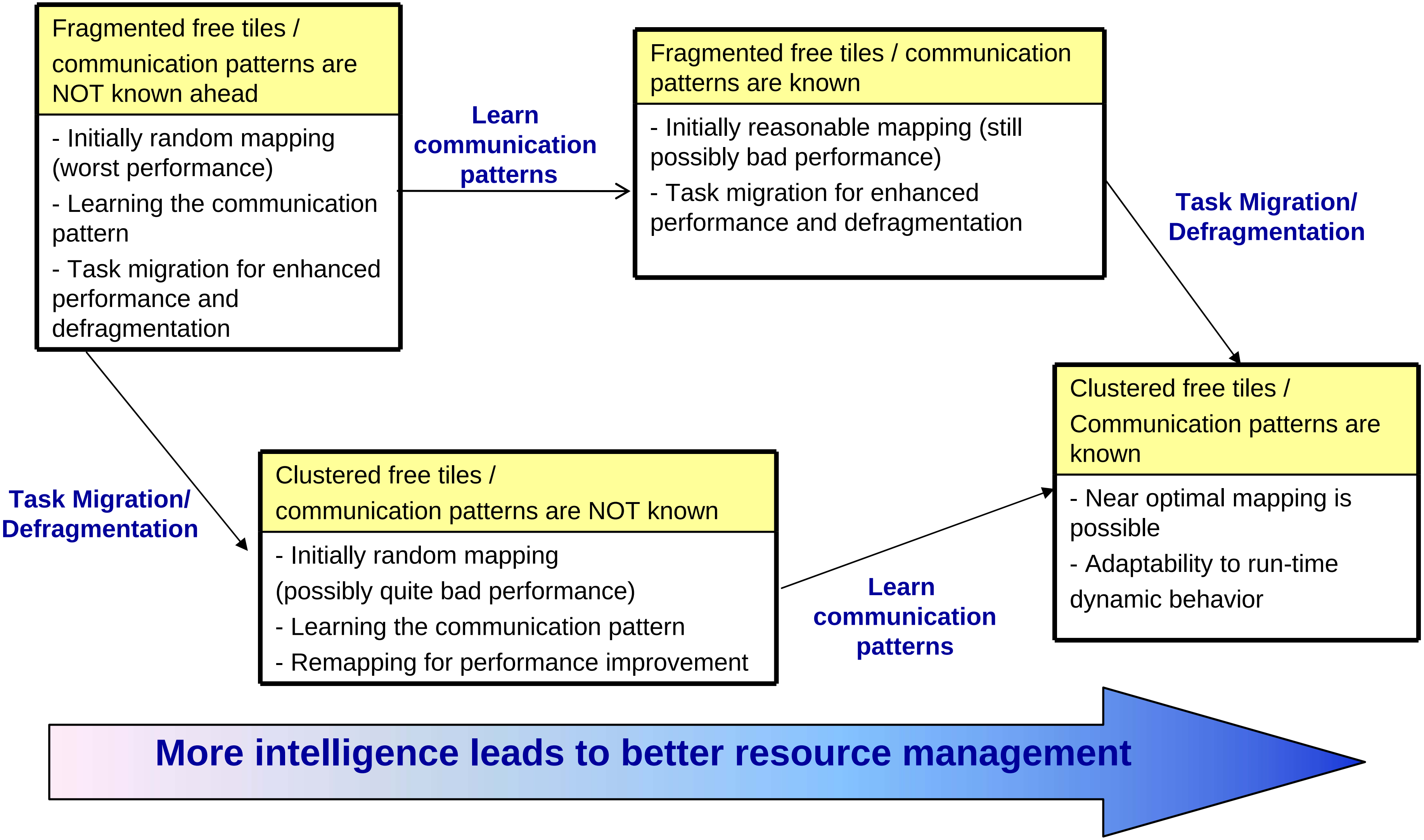
With higher GMTI allocation and task frequencies (performance), random mapping does not work at all.

Outline

- Introduction
- Simulation Model
- Applications and Results
  - □ Resource Fragmentation
  - □ Mapping Performance
- Problem Space and Future Work
- Conclusion

Approved for public release, distribution unlimited

Network-Oriented Problem Space



Fragmented free tiles / communication patterns are NOT known ahead

- Initially random mapping (worst performance)
- Learning the communication pattern
- Task migration for enhanced performance and defragmentation

Learn communication patterns

Fragmented free tiles / communication patterns are known

- Initially reasonable mapping (still possibly bad performance)
- Task migration for enhanced performance and defragmentation

Task Migration/ Defragmentation

Clustered free tiles / communication patterns are NOT known

- Initially random mapping (possibly quite bad performance)
- Learning the communication pattern
- Remapping for performance improvement

Learn communication patterns

Clustered free tiles / Communication patterns are known

- Near optimal mapping is possible
- Adaptability to run-time dynamic behavior

More intelligence leads to better resource management

Future Work

- Run-time profiling technique
  - □ Low overhead
  - □ Detecting network load imbalance
  - □ Detecting communication patterns, dependencies
  - □ Detecting processor load imbalance
- Gradual morphing technique
  - □ Intelligent partial remapping of the application
    - Hot spot removal, load balancing
  - □ Adaptive changes of parallelism of the application
    - Adaptive to the status of free resources
- Run-time support for gradual morphing
  - □ Task migration
  - □ Dynamic changes of parallelism of the application
- Dynamic adaptation of code
  - □ Expression of dynamic changes of code at run-time
  - □ Dynamic code generation

Approved for public release, distribution unlimited

Conclusion

- Large multi-tiled architectures will require intelligent run-time systems for resource allocation and dynamic mapping
  - □ Future applications will be complex and will need to respond to unexpected events
  - □ Future, large-scale multi-tiled architectures will require application sharing and arbitration of resources
- In this research, we have explored the effect of dynamic application variations on multi-tiled performance
  - □ Dynamic IRT with representative mode changes and load variations causes resource fragmentation over time
  - □ Application mappings must be done carefully in order to provide robust behavior
- Future work will involve the application of knowledge-based and machine learning algorithms to the profile-based dynamic resource management

Approved for public release, distribution unlimited