計算状態操作機構による並列言語実装と評価

研究代表者：八杉 昌宏（九州工業大学）
(基盤研究(B)「計算状態の精密操作に基づく
高性能・高信頼システム技術」
研究分担者：平石 拓（京都大学）
研究分担者：光来 健一（九州工業大学））

計算状態の精密操作に基づく高性能・高信頼システム技術

- 科研費基盤(B) 2014年度より
  - 研究代表者　八杉　昌宏　(九州工業大学)
  - 研究分担者　平石　拓　(京都大学)
  - 研究分担者　光来　健一　(九州工業大学)
  - 連携研究者　小出　洋　(九州工業大学)
  - 連携研究者　江本　健斗　(九州工業大学)
- 2013年度まで
  - 科研費基盤(B): 安全な計算状態操作機構の実用化

発表内容

- (安全な)計算状態操作機構L-closureとは?
  - AT との関係
- これまでの応用例 (L-closureで実現)
- これまでの実装 (L-closureを実現)
- 最近の応用例
- 最近の実装
- これからの課題

L-closure: 計算状態操作機構とは? (1/2)

- 高水準言語のコンパイラの中間言語で提供される言語機構として提案
  - 高信頼，高性能な高水準言語の実装
- 実行中のソフトウェアの動的再構成・保全
  - 呼び出し元で眠っている変数への合法的アクセス
  - 汎用性が高く，さまざまに利用可能
  - ごみ集め，真の末尾再帰，動的負荷分散など
  - 表面的ではなく抜本的に途中で見直せる
  - 自分自身を対象としたデバッガを操作するように

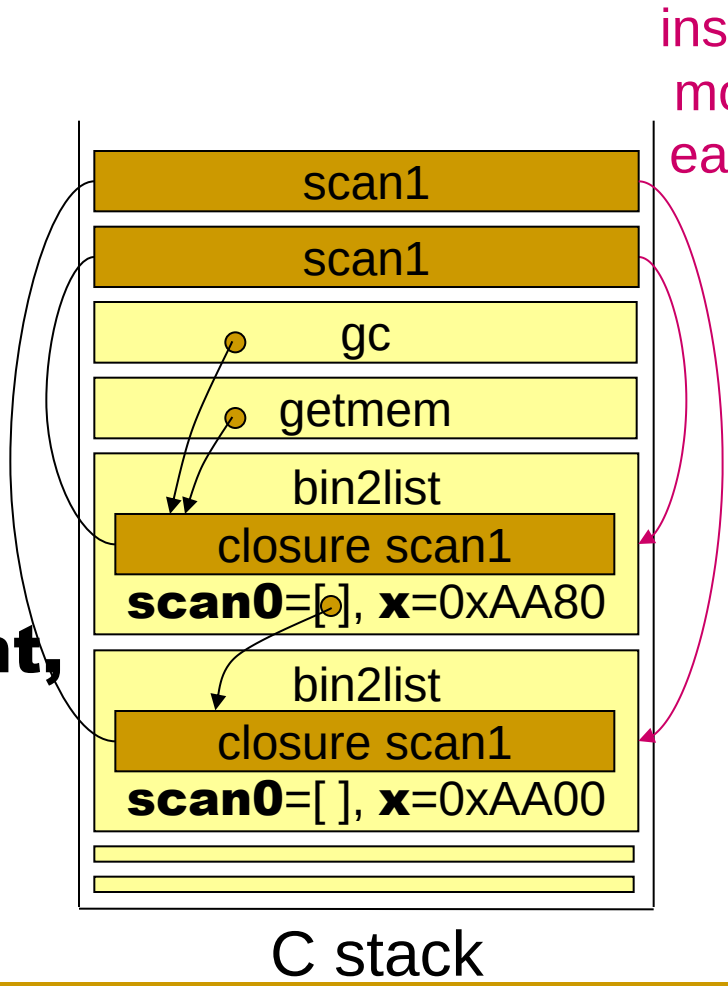
L-closure: 計算状態操作機構とは? (2/2)

- 拡張C言語の仕様として
- 入れ子関数(クロージャ)を利用
  - 呼出し元に眠る変数の値への安全で正式なアクセス
- 高度な技法で高性能実装
  - アクセス対象となる変数もレジスタ割り当て候補
  - 実際に呼び出すまでクロージャの初期化を遅延
  - 主要な2つの実装
    - GCC拡張による実装 [八杉 他 CC2006, IPSJ PRO論文誌2008 (平成21年度論文賞)]
    - 標準C言語への翻訳方式 [平石・八杉他 IPSJ PRO論文誌2006] [田附・八杉・平石他 IPSJ PRO論文誌2013]

```
Alist *bin2list(void (*scan0) closure (move_f),
                Bintree *x, Alist *rest){
  Alist *a = 0; KVpair *kv = 0;
 void scan1 closure (move_f mv){  /* create closure */
   x = mv(x); rest = mv(rest);    /* scan roots */
   a = mv(a); kv = mv(kv);        /* scan roots */
   scan0(mv);                     /* scan older roots */
  }
 // pass pointer to closure "scan1" on the following calls.
  if(x->right) rest = bin2list(scan1, x->right, rest);
  kv = getmem(scan1, &KVpair_d);    /* allocation */
  kv->key = x->key; kv->val = x->val;
  a = getmem(scan1, &Alist_d);      /* allocation */
  a->kv = kv; a->cdr = rest;
  rest = a;
  if(x->left) rest = bin2list(scan1, x->left, rest);
  return rest;
}
```

Using Lexical Closures

```
/* simplified for explanation */
bin2list(closure_t scan0,
     Bintree *x, ...){
     void scan1 closure (){
          x = move(x);
          scan0();
     }
     bin2list(scan1, x->right,
     ...);
     getmem(scan1,... );
}
```



実行時(抜本的)細部選択の自由 (1/2)

- 同じ計算内容で複数手段
  - 負荷分散: ある計算を逐次実行? 並列実行? いつ, どこで実行?
- 期待できる最良の選択は?
  - 局所的観点からは「逐次実行」を選択
  - それればかりだと, 並列実行の機会なし → 「合成の誤謬」的な事態に

実行時(抜本的)細部選択の自由 (2/2)

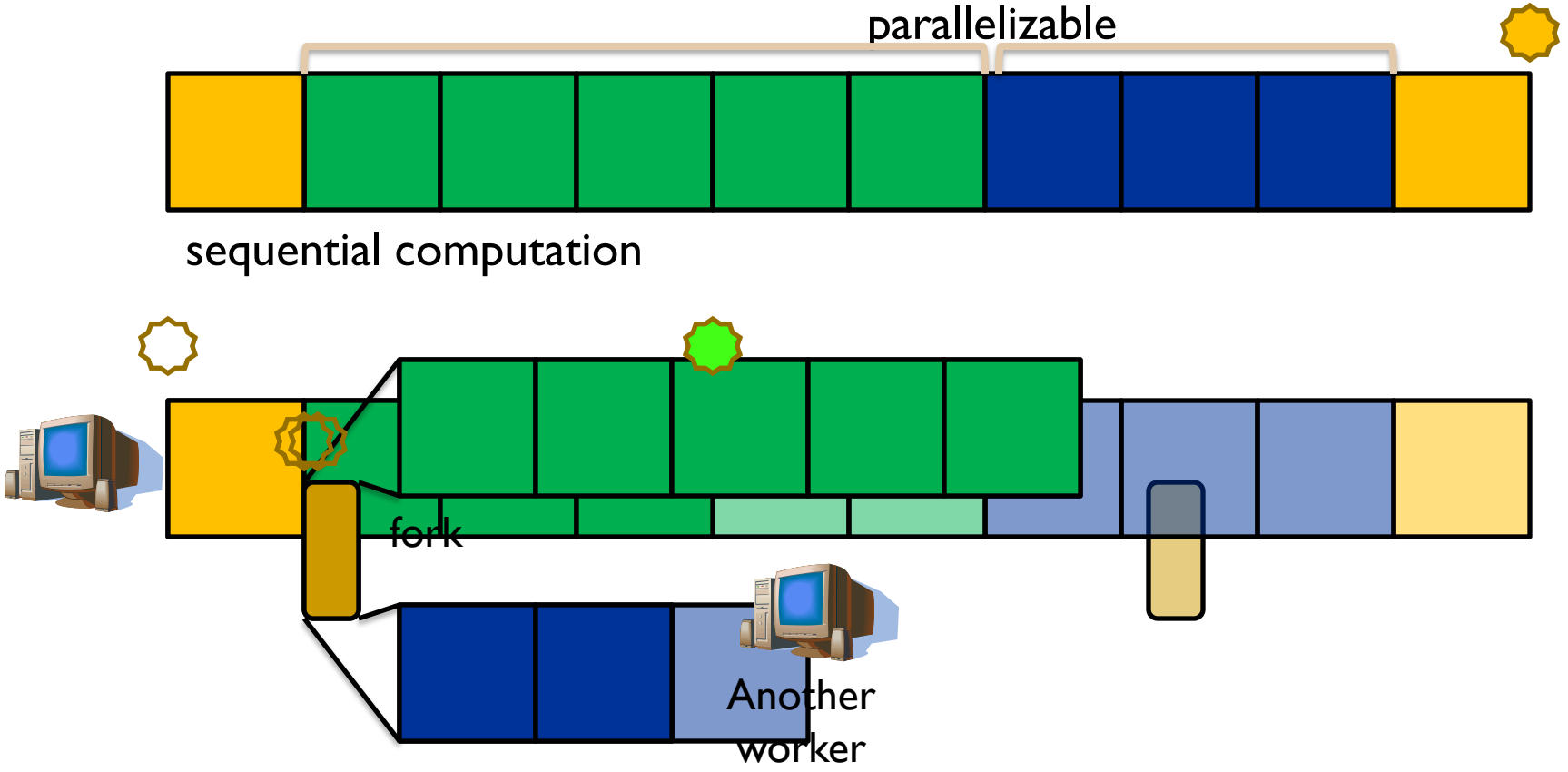
- ある種のジレンマ
  - あるrunで，常に同じ選択ではいけない
- ゲーム理論の混合戦略(ある確率で選択)は?
  - 選択すべき時にはまだ情報不足
- 後から選択を変更できるとよいが?
  - 計算状態操作機構で可能
    (呼び出し元の変数の値の参照・変更などにより)
- 負荷分散の例
  - 逐次実行を後から並列実行に変更

発表内容

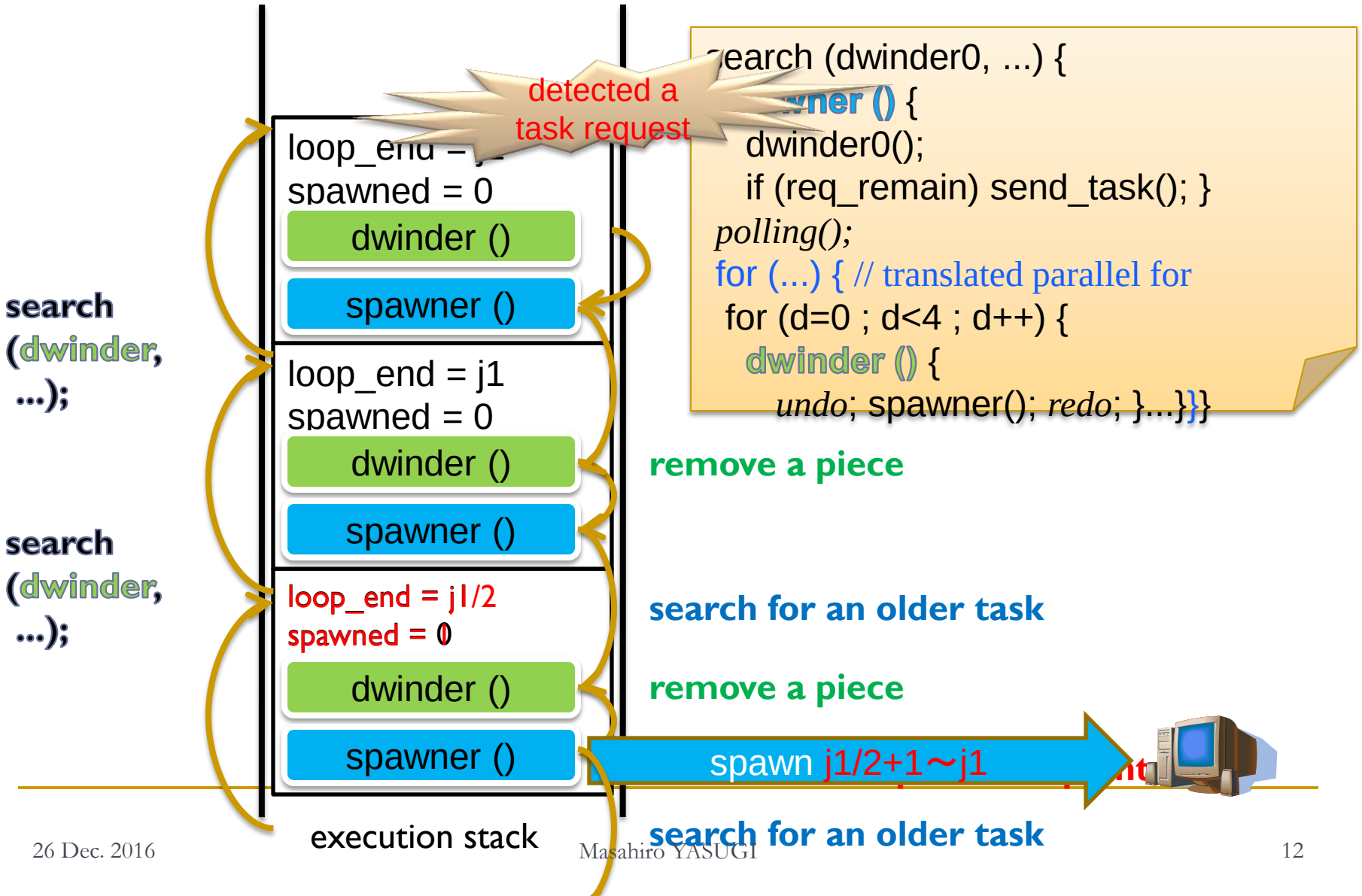
- (安全な)計算状態操作機構L-closureとは?
  - AT との関係
- これまでの応用例
  - Tascell: Backtracking-based Load Balancing [平石・八杉 他 PPoPP 2009]
  - 真の末尾再帰 (proper tail recursion) [ILC 2010]
  - Tascell: *N*体問題（Barnes-Hut algorithm）[SACSIS 2012]
  - Tascell: 共通アイテム集合を持つ連結部分グラフ抽出の並列化 [JSIAM2013, IPSJ PRO論文誌2014]
  - Tascell に例外処理を導入 [SWoPP-PRO 2015]
    - 共通アイテム集合を持つ連結部分グラフ抽出での探索打ち切りへの適用 [ACSI2015 (Outstanding Research Award)]
  - 仮想環境におけるTascell の動作の分析 [ISPA 2015]
- これまでの実装
- 最近の応用例
- 最近の実装
- これからの課題

Tascell: Load Balancing Framework [PPoPP2009]

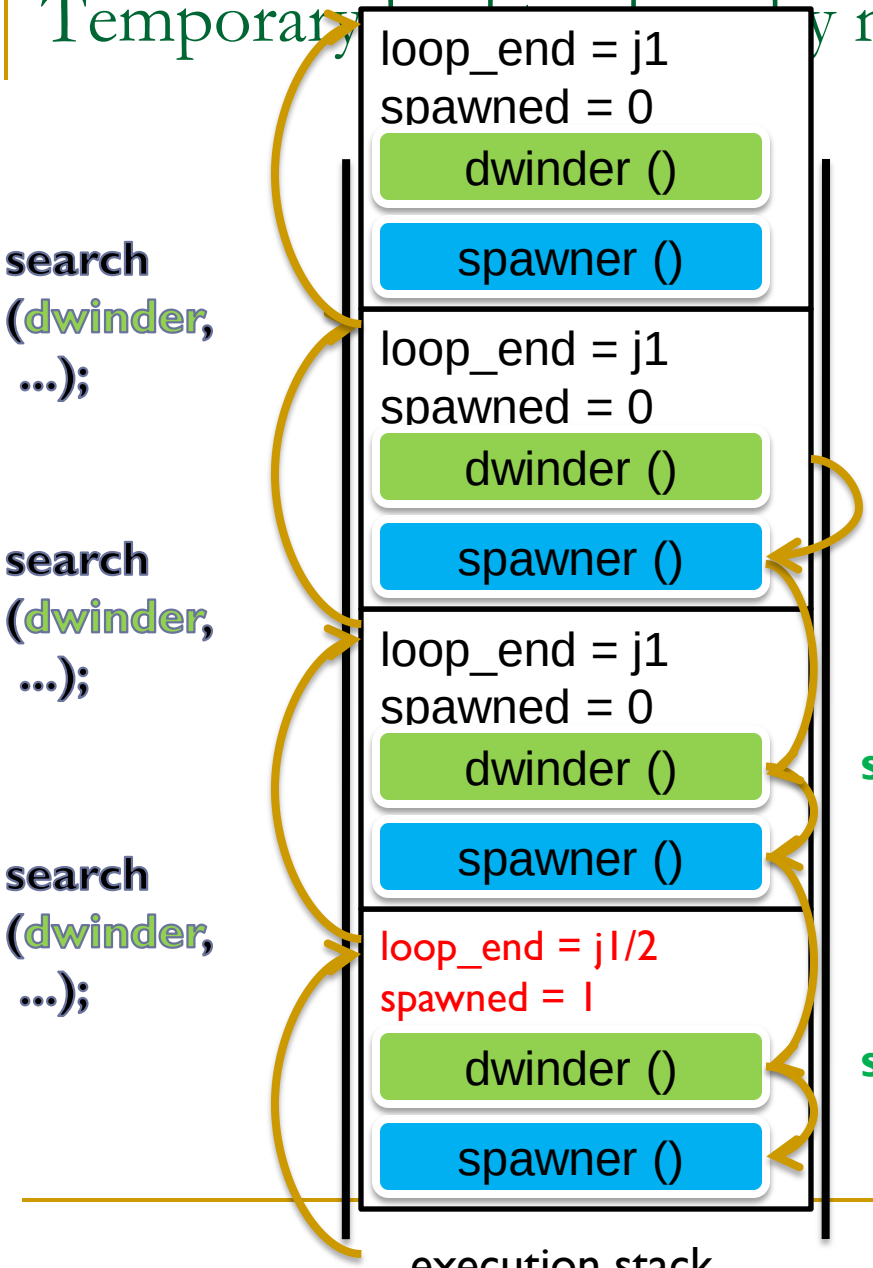
- *"Logical thread"-free*
- Compute sequentially, fork only when requested



Temporary backtracking by nested functions



Temporary [illegible] by nested functions

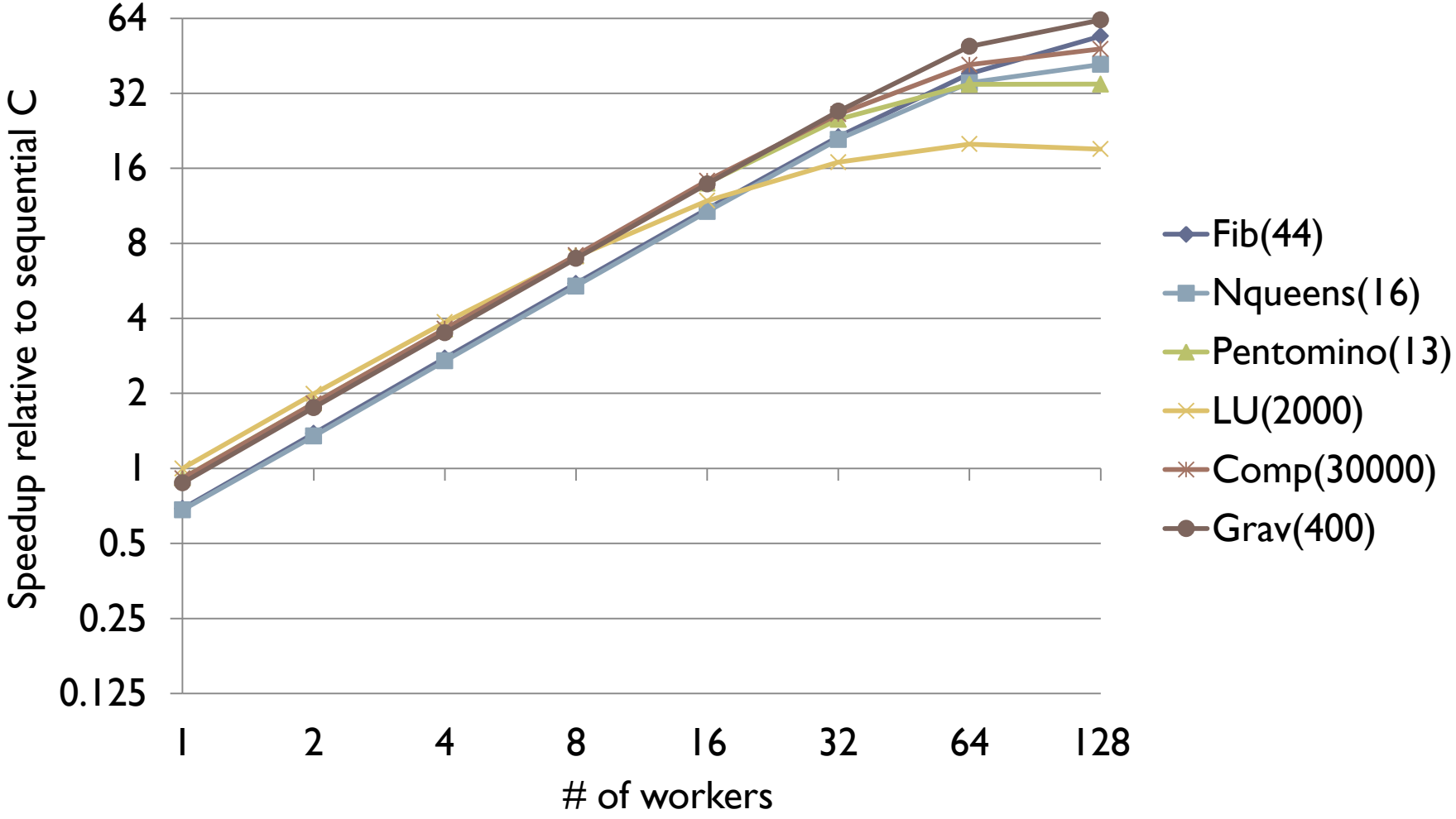



```
search (dwinder0, ...) {
 spawner () {
   dwinder0();
   if (req_remain) send_task(); }
 polling();
 for (...) { // translated parallel for
  for (d=0 ; d<4 ; d++) {
    dwinder () {
      undo; spawner(); redo; }...}}}
```

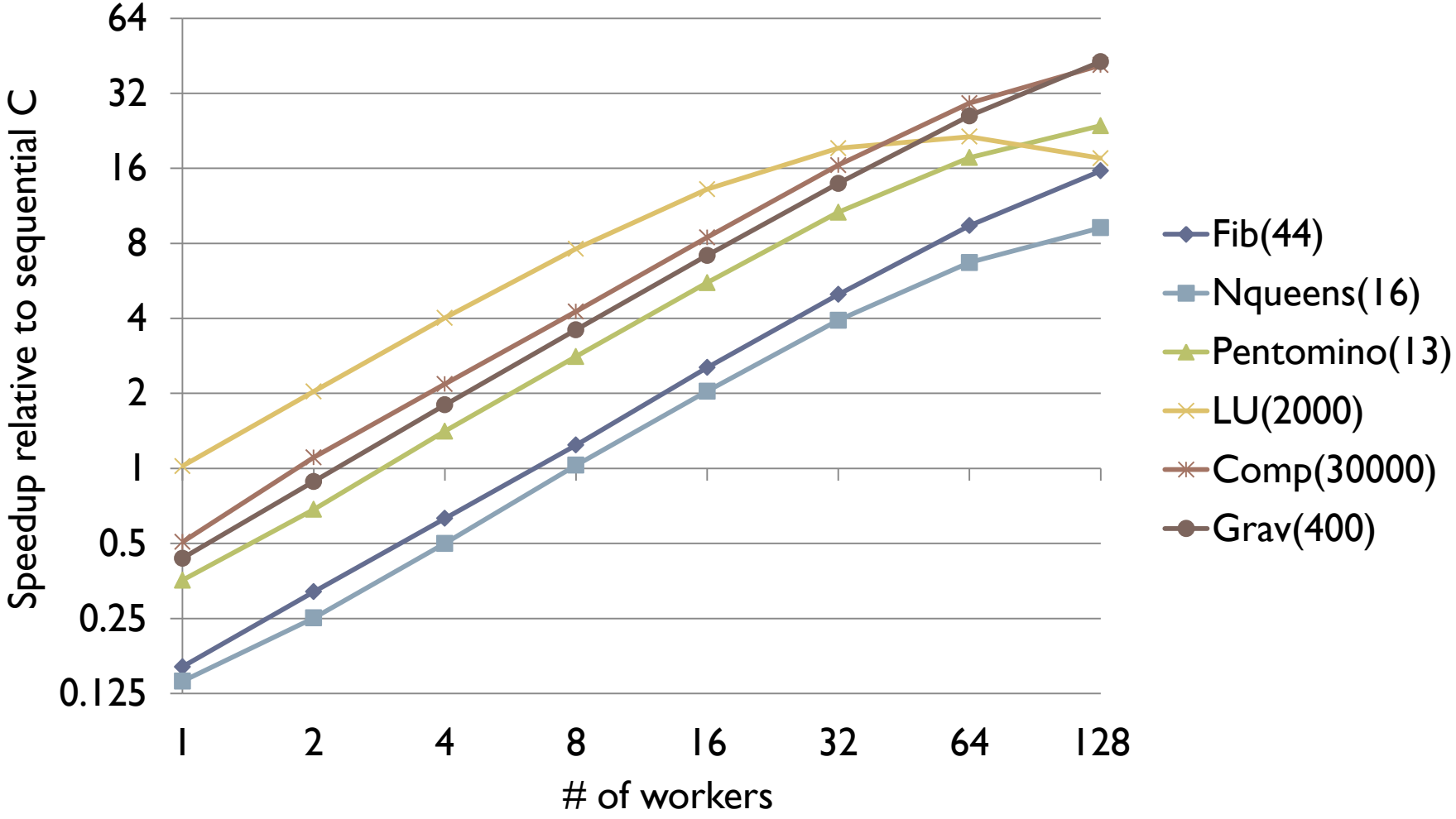
set the removed piece

set the removed piece

測定結果（Niagara2 Tascell）



測定結果（Niagara2 Cilk [PLDI'98]）



発表内容

- (安全な)計算状態操作機構L-closureとは?
  - AT との関係
- これまでの応用例
- これまでの実装 (L-closureを実現)
  - GCC拡張の実装モデル
    - GCC 3.4.6ベース, SPARC, IA-32, x86-64
    - GCC バージョン4 ベースへ，ただし(lazyではない) closure の実装
  - 標準C言語への翻訳 (LW-SC)
    - 明示的スタックの利用
    - さらに，償却時間(amortized time)を下げる実装モデル
- 最近の応用例
- 最近の実装
- これからの課題

今後の課題

- GCC4へのL-closureの実装

		コンパイラ			変換
		Trampoline	Closure	L-closure	LW-SC
GCC3	old	○	○	○	○
	new	-	-	必要？	●
GCC4	old	○	●	×	○
	new	-	-	難しい	●
インテルコンパイラ LLVM 等		○	未定	未定	●

発表内容

- (安全な)計算状態操作機構L-closureとは?
  - AT との関係
- これまでの応用例
- これまでの実装
- 最近の応用例
  - Tascell: 分散メモリ環境における共通アイテム集合を持つ連結部分グラフ抽出の並列化 [IPSJ-PRO研究会 2016]
  - Tascell: MPIベース実装，大規模並列環境での評価 [P2S2 2016]
  - Tascell: 確率的ガード [P2S2 2016]
  - 仮想環境における資源不足時のTascellの性能低下の確認 [JSSST大会 2016]
- 最近の実装
- これからの課題

Evaluation of an MPI-based Implementation of the Tascell Task-Parallel Language on Massively Parallel Systems

August 16, 2016

Daisuke Muraoka, Kyushu Institute of Technology
Masahiro Yasugi, Kyushu Institute of Technology
Tasuku Hiraishi, Kyoto University
Seiji Umatani, Kyoto University

Background

- We develop the Tascell task parallel language
- A Tascell program can be executed efficiently both in shared-memory and distributed-memory environments
- The conventional Tascell realizes inter-node communication with TCP/IP
  - Cannot run on supercomputers in which TCP/IP is not available for inter-node communication

Contributions

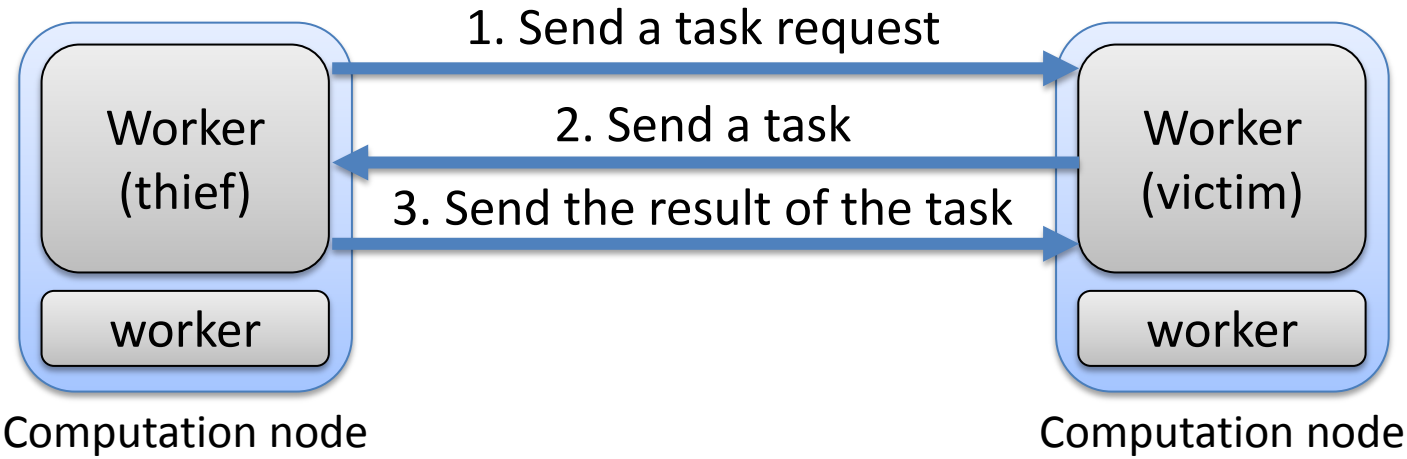
- Implement inter-node communication using MPI
- Evaluate the performance on the K computer using 7168 workers (1024 nodes)
  - Our 19-queens solver achieves a 4615-fold speedup relative to a serial C program
- Compare our MPI-based implementation with other implementations

Outline

- The conventional Tascell
- Our MPI-based implementation
- Evaluations
- Related work
- Conclusions

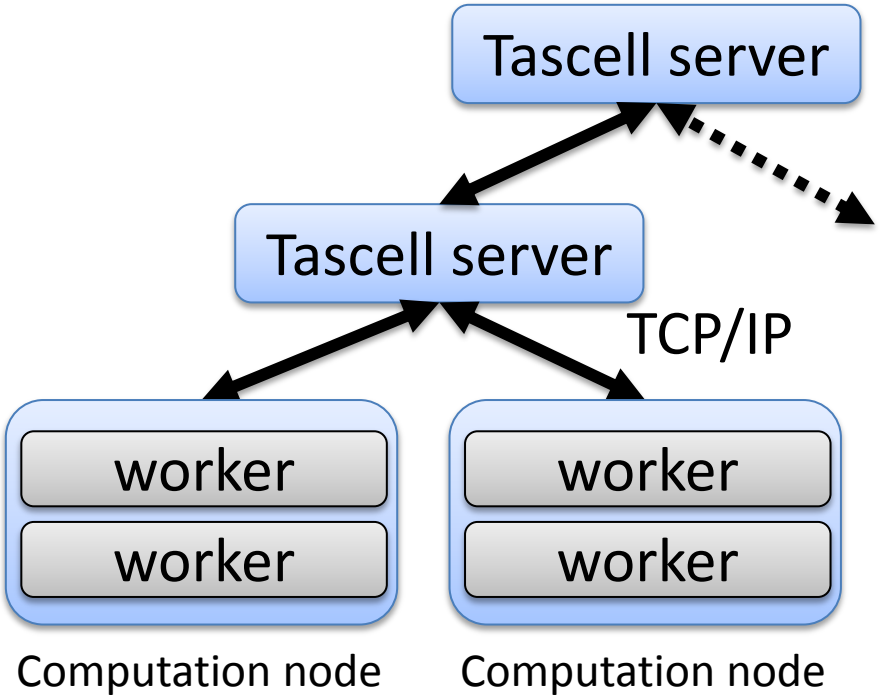
Tascell

- Tascell adopts work-stealing
  1. An idle worker(thief) sends a task request to any worker(victim)
  2. The victim sends a task as a response
  3. When the thief completes the stolen task, it sends the result to the victim worker



Tascell server

- Determines the destination of each task request
- Knows which computation nodes have tasks



☺ Suitable for wide-area distributed environments

☺ Computation nodes can be added dynamically

☹ Tascell server becomes a bottleneck

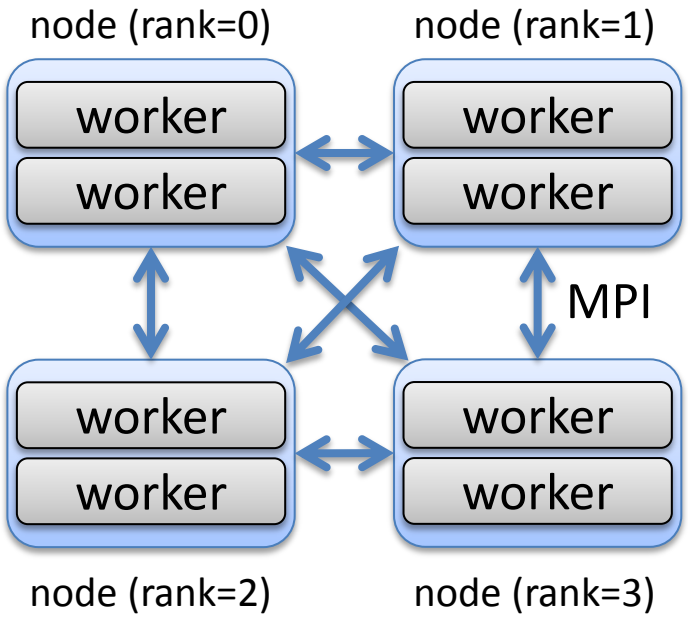
☹ Some supercomputers do not support TCP/IP

Outline

- The conventional Tascell
- Our MPI-based implementation
- Evaluations
- Related work
- Conclusions

Our MPI-based implementation

- "Tascell server"-less
- Inter-node communication with MPI
  - Requires only MPI_THREAD_FUNNELED support level



MPI multithread support levels

- MPI_THREAD_SINGLE
  - Only a single thread can run
- MPI_THREAD_FUNNELED
  - Only the thread that has called MPI_Init_thread can make MPI calls, and other
- MPI_THREAD_SERIALIZE
  - Only one thread can make
- **MPI_THREAD_MULTIPLE**
  - **Multiple threads may make MPI calls concurrently**

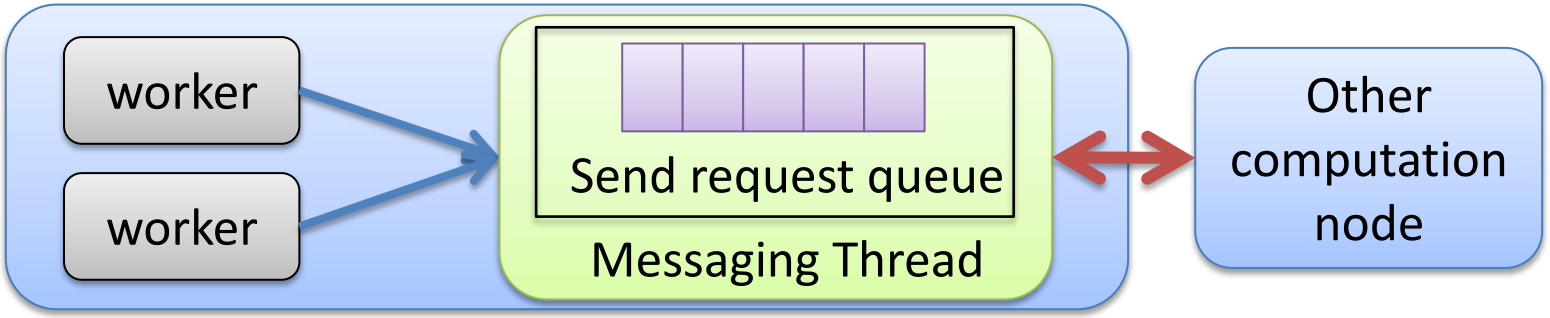
Unsupported on the K computer

MPI-based implementation issue

- Work stealing
  - Cannot statically resolve dynamically determined wait-for dependencies
- Cannot implement the naïve idea
  - Only possible with `MPI_THREAD_MULTIPLE`
  - Many environments do not support or recommend `MPI_THREAD_MULTIPLE`

Our MPI-based implementation

- Workers add an entry to the send request queue
- Only the messaging thread makes MPI calls
- Send a message using a nonblocking function
- Checks whether there are any incoming message using MPI_Iprobe



Messaging thread

Repeat the following steps:

1. Checks whether there are any incoming message using MPI_Iprobe
   a. And if any, receives a message using MPI_Recv
2. If there are any entries in a send request queue, sends a message using MPI_Isend
3. Checks whether MPI_Isend operation is complete using MPI_Test

Deadlock avoidance

- We use busy waiting in order to wait for the following events at the same time in a single thread
  - Completion of send
  - Receiving messages
  - Add an entry to the send request queue
- We cannot avoid deadlocks without busy waiting because we only require the MPI_THREAD_SERIALIZED support level

Outline

- The conventional Tascell
- Our MPI-based implementation
- Evaluations
- Related work
- Conclusions

Evaluations

	Xeon Phi	The K computer (per node)
Host processor	Xeon E5-2697 v2 12-core × 2	SPARC64 VIIIfx 2GHz 8-core
Coprocessor	Xeon Phi 3120P 57-core × 4	-
Network	PCIe3.0 x16	Tofu Interconnect
MPI library	Intel MPI 4.1 Update 3	FujitsuMPI（OpenMPI based）

- Load balancing benchmarks:
  - Fibonacci, Fib(*n*)
  - N-Queen problem, Nq(*n*)
  - Pentomino problem, Pen(*n*)
  - AreaSum, AreaSum(*n*)

  } UTS-like search trees

  - Matrix Multiplication, MatMul(*n*)
  - LU decomposition, LU(*n*)

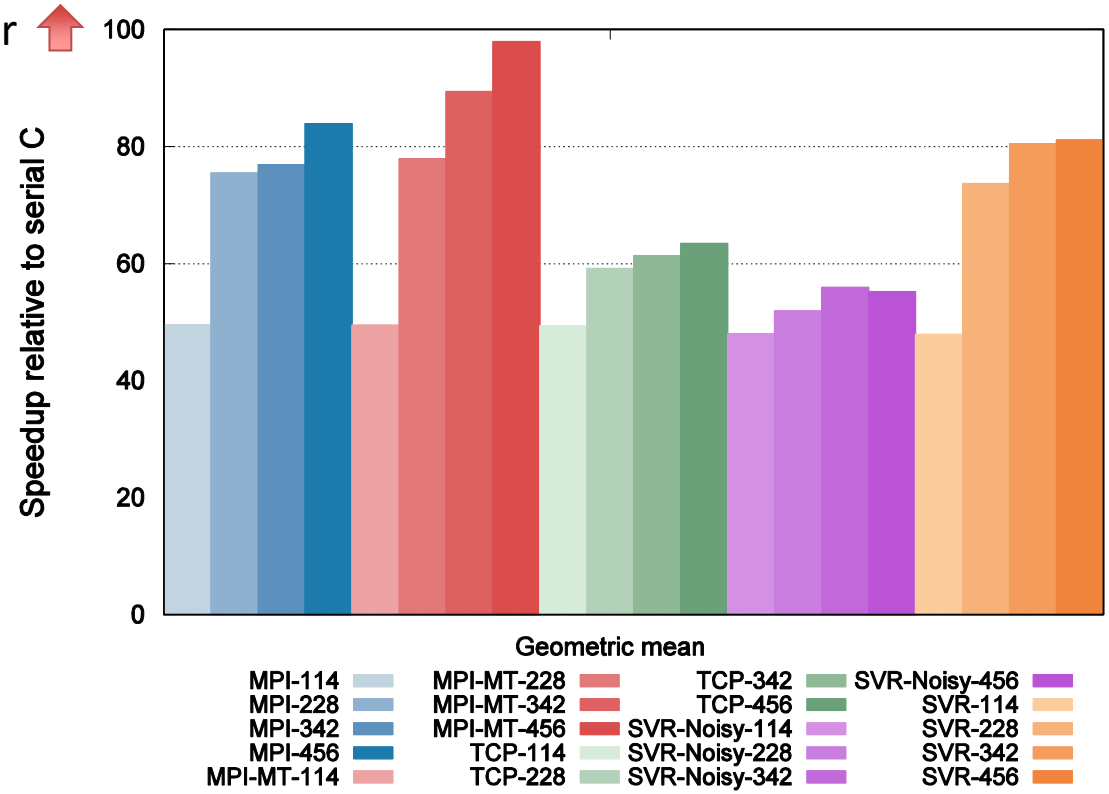
Implementations in evaluations

	Inter-node communication	Tascell server	Run on the K computer
Tascell/MPI (Our MPI-based implementation)	MPI (MPI_THREAD_FUNNELED)	None	Yes
Tascell/MPI-MT	MPI (MPI_THREAD_MULTIPLE)	None	No
Tascell/TCP	TCP/IP	None	No
Tascell/SVR-Noisy	TCP/IP	Require (relay only)	No
Tascell/SVR (The conventional implementation)	TCP/IP	Require (knows which nodes have tasks)	No

Evaluations on Xeon Phi:

Small benchmarks with geometric mean except LU

More is better

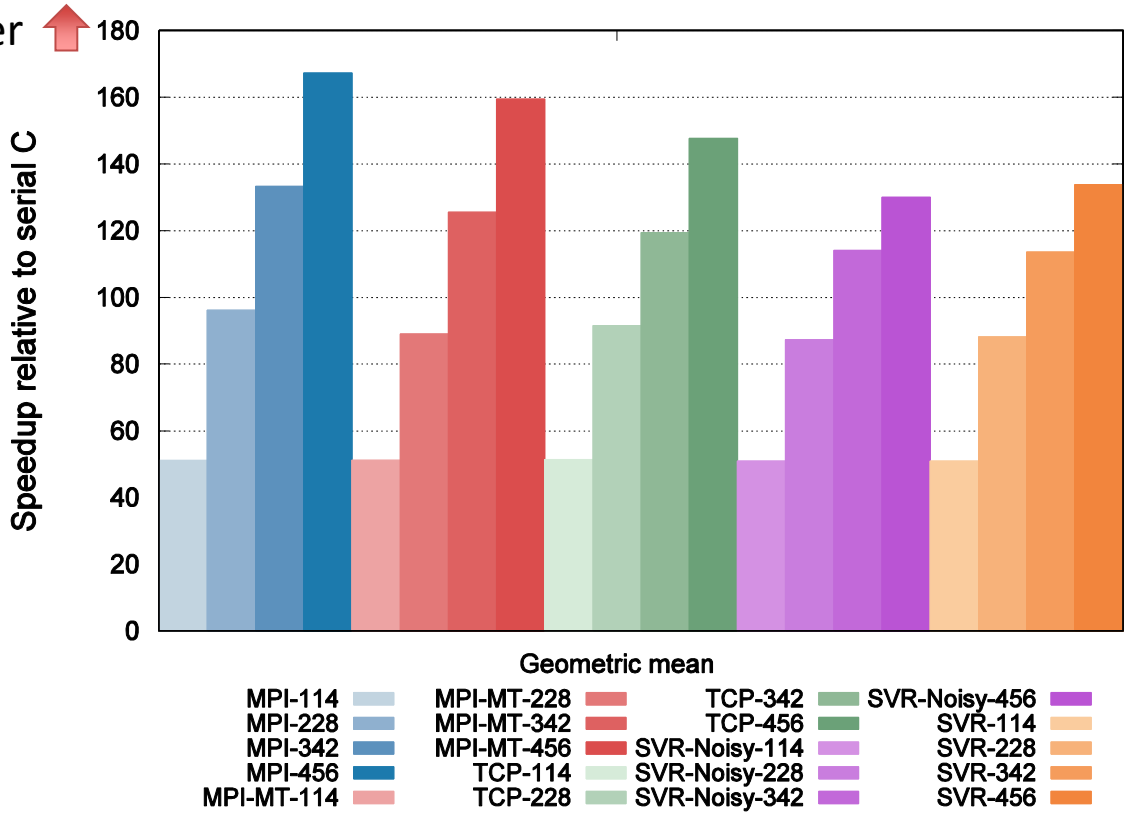


Speedup relative to serial C.
Geometric mean over Fib(48), Nq(16), Pen(14), AreaSum(18) and MatMul(7000).
228: 2 nodes x (114 workers/node)
342: 3 nodes x (114 workers/node)
456: 4 nodes x (114 workers/node)

Evaluations on Xeon Phi:

Large benchmarks with geometric mean except LU

More is better

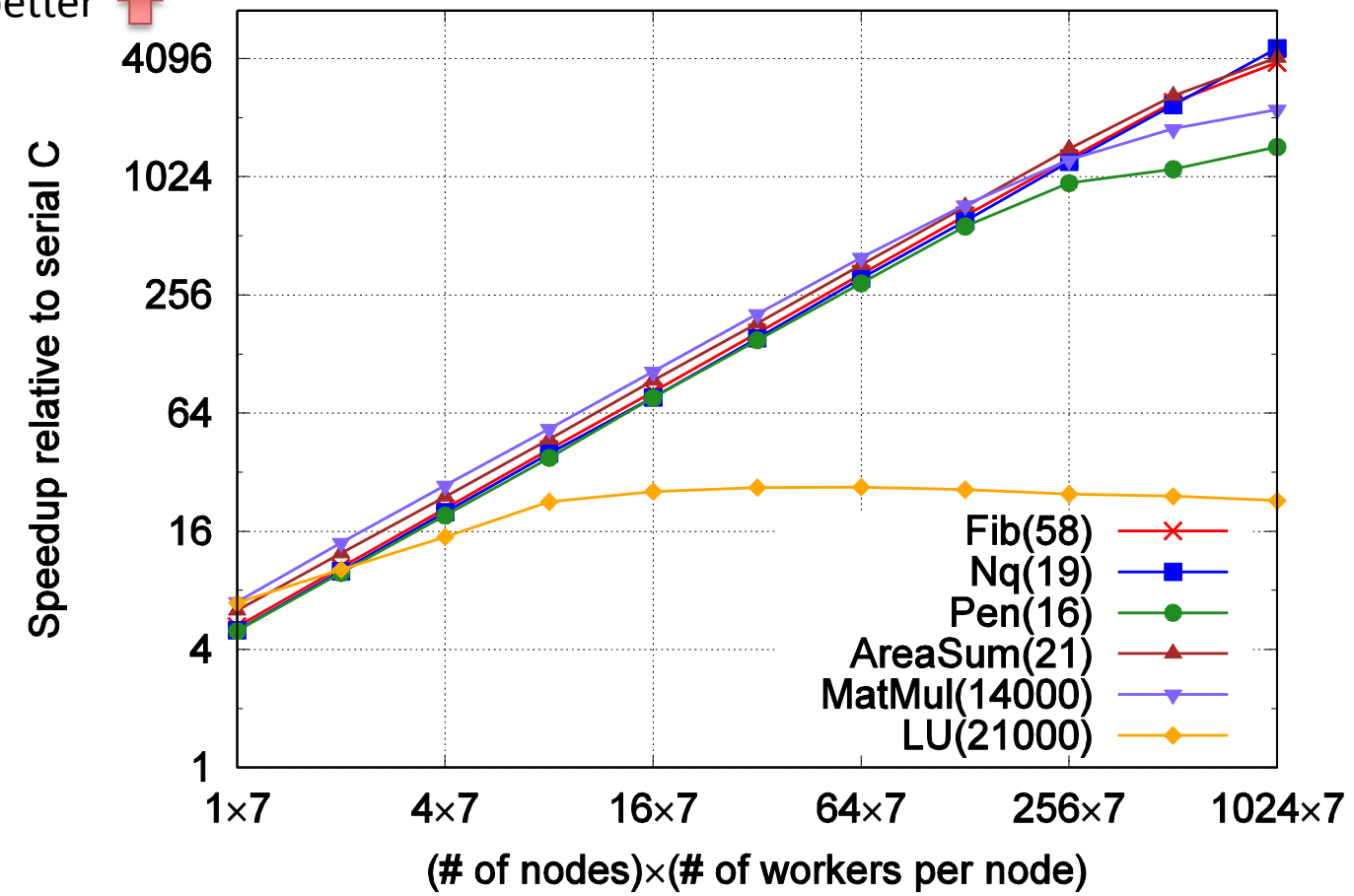


Speedup relative to serial C.
Geometric mean over Fib(58), Nq(19), Pen(16), AreaSum(21) and MatMul(14000).
228: 2 nodes x (114 workers/node)
342: 3 nodes x (114 workers/node)
456: 4 nodes x (114 workers/node)

Evaluations on the K computer:

Large benchmarks

More is better



Speedups relative to serial C

Conclusions

- Implement inter-node communication for the Tascell task parallel language using MPI
  - It requires only the MPI_THREAD_FUNNELED support level
- Evaluate the performance on the K computer using 7168 workers (1024 nodes)
- Compare our MPI-based implementation with other implementations on Xeon Phi coprocessors
  - Tascell/SVR is better than Tascell/SVR-Noisy
  - MPI-based implementations outperform other implementations

発表内容

- (安全な)計算状態操作機構L-closureとは?
  - AT との関係
- これまでの応用例
- これまでの実装
- 最近の応用例
  - Tascell: 分散メモリ環境における共通アイテム集合を持つ連結部分グラフ抽出の並列化 [IPSJ-PRO研究会 2016]
  - Tascell: MPIベース実装，大規模並列環境での評価 [P2S2 2016]
  - Tascell: 確率的ガード [P2S2 2016]
  - 仮想環境における資源不足時のTascellの性能低下の確認 [JSSST大会 2016]
- 最近の実装
- これからの課題

Extending a Work-Stealing Framework with Probabilistic Guards

Aug. 16, 2016

Hiroshi Yoritaka†, Ken Matsui‡§, Masahiro Yasugi†, Tasuku Hiraishi‡ and Seiji Umatani‡

†Kyushu Institute of Technology, ‡Kyoto University, §Presently with Nintendo Co., Ltd.

Contributions

- Propose probabilistic guards (PG) and virtual probabilistic guards (VPG)
  - New mechanisms for work-stealing frameworks
  - Prevent thieves from stealing small tasks from victims probabilistically
  - Aim to reduce the total task division cost
  - PG can skew the probabilities of eventual success in repeated uniformly random steal attempts
- Implement our proposals on a work-stealing framework
- Performance evaluation with Barnes-Hut algorithm

Outline

Contributions

Background

Tascell

Our proposals

Probabilistic guards

Virtual probabilistic guards

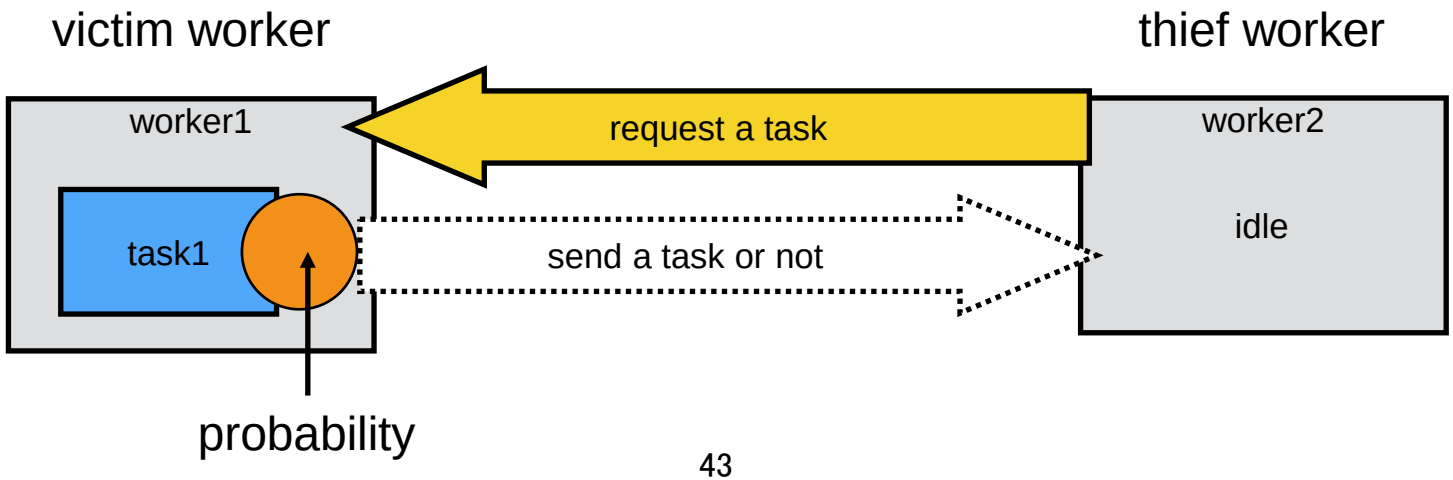
Evaluations

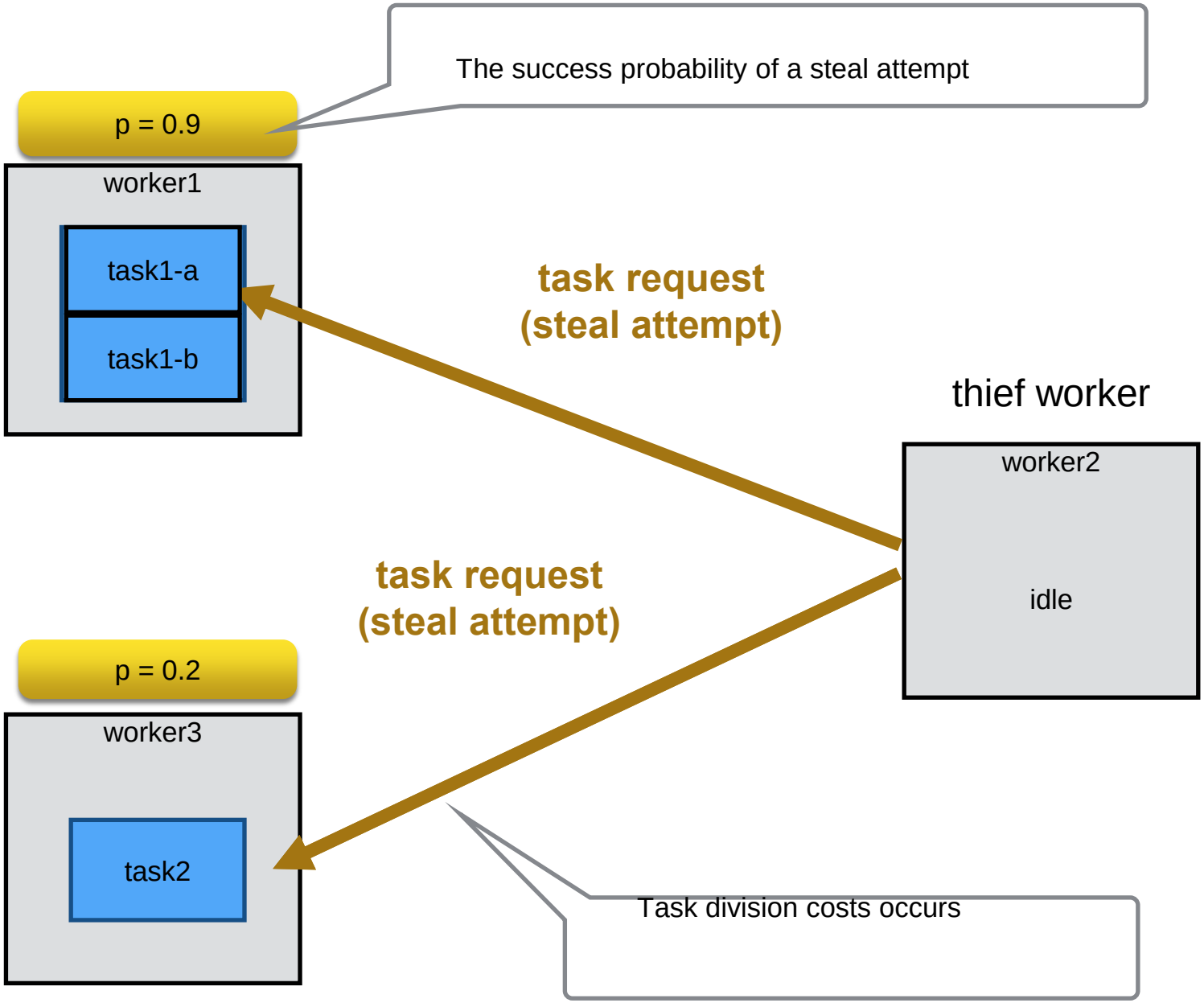
Barnes-Hut algorithm

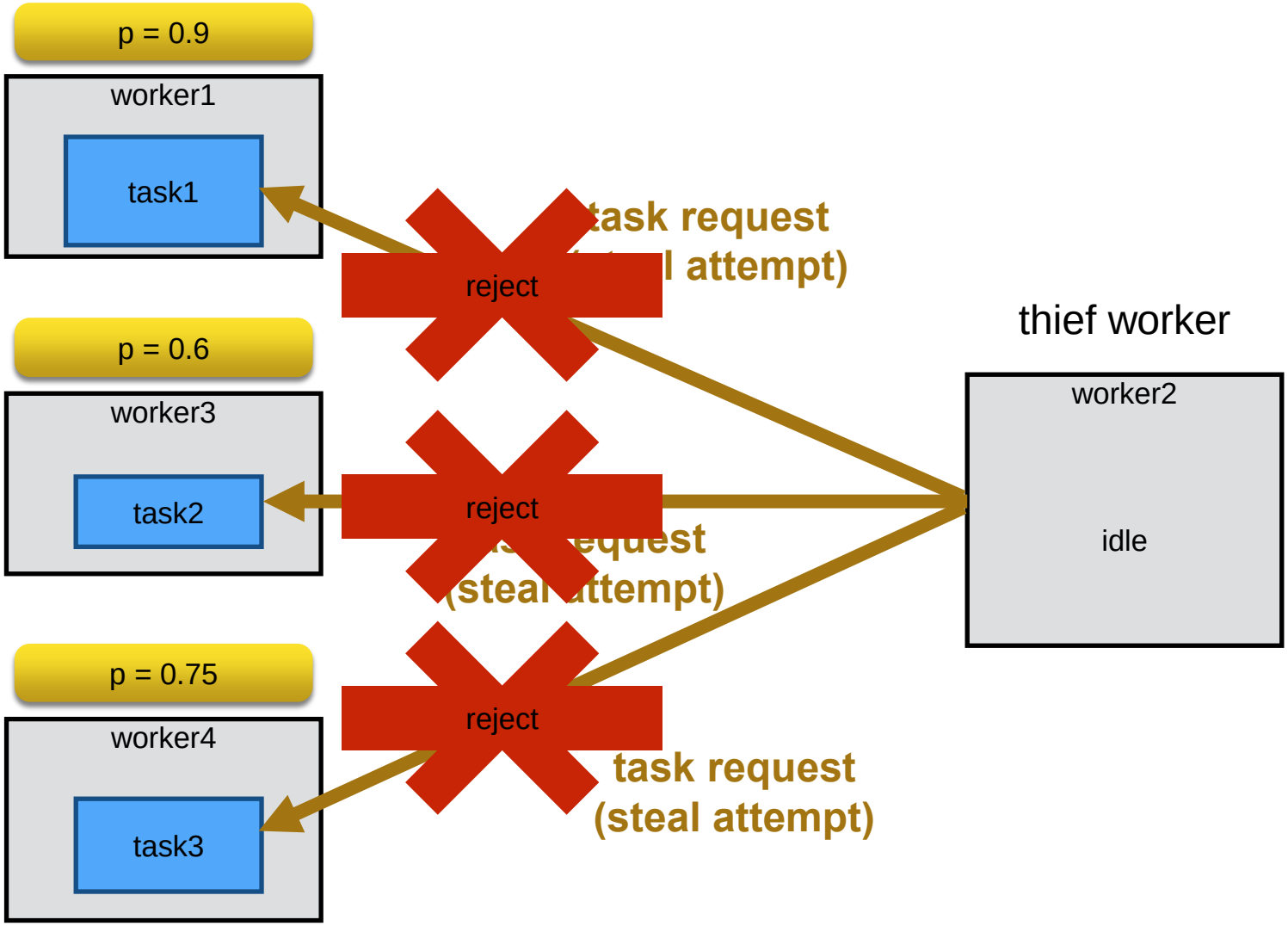
Experimental results

Probabilistic guards (PG)

- A mechanism to reduce the total task division costs
- Prevents thieves from stealing small tasks from victims probabilistically
- Can skew the probabilities of eventual success in repeated uniformly random steal attempts
- Programmers can set a probability that tasks of a worker can be stolen at runtime



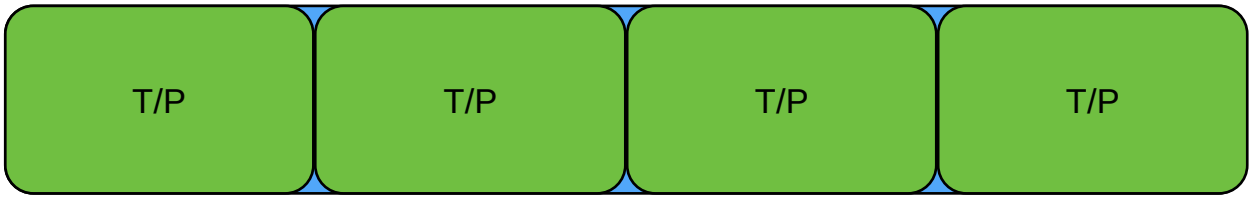

The success probability of a steal attempt
p = 0.9
worker1
task1-a
task1-b
task request
(steal attempt)
thief worker
worker2
idle
task request
(steal attempt)
p = 0.2
worker3
task2
Task division costs occurs


p = 0.9
worker1
task1
reject
task request
l attempt)
thief worker
p = 0.6
worker3
task2
reject
worker2
idle
equest
(steal attempt)
p = 0.75
worker4
reject
task request
(steal attempt)
task3

Upper limit for probabilistic guards

- Limits the number of repeated probabilistically prevented steal attempts
- To avoid an unbounded number of prevented steal attempts

The total task : *T*



An ideal *Ti* (amount of work of the *i*-th worker)

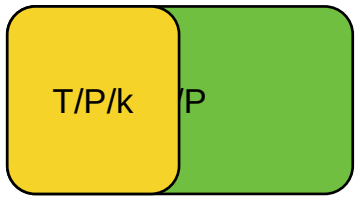
→ Determine whether steal attempts succeed

by *Ti* > *T*/*P*/*k* (that is $\frac{k * P * T_i}{T} > 1$

defining k = 3 to 25 would be effective to reduce total load imbalance to 4 to 33%

The steal success probability :

$$MIN(1.0, \frac{k * P * T_i}{T})$$



k = 2 : *Ti* = T/P/k,

The steal success probability = 1.0,
PG start to work

k = 2 : *Ti* = T/P/k/2

The steal success probability = 0.5
→ easy to induced to larger tasks

Summary of probabilistic guards

- Prevent thieves from stealing small tasks from victims probabilistically
- Aim to induce the steal attempts to larger tasks
  - Reduce the total task division cost
- Can skew the probabilities of eventual success in repeated uniformly random steal attempts
- With an upper limit, a thief will not repeat an unbounded number of probabilistically prevented steal attempts
- Programmers can set a probability that tasks of a worker can be stolen
- Potential concurrency is not lost unlike threshold-based granularity control

Outline

Contributions

Background

Tascell

Our proposals

Probabilistic guards

Virtual probabilistic guards

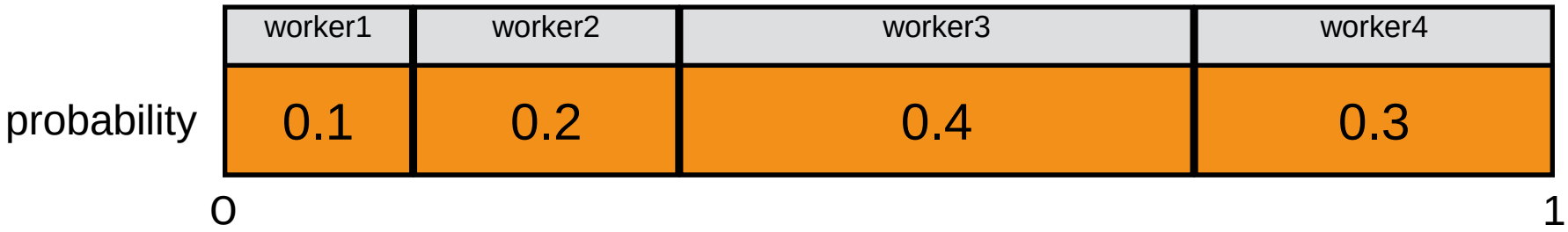
Evaluations

Barnes-Hut algorithm

Experimental results

Virtual probabilistic guards (VPG)

- Thief checks the probabilities of all victims
- Select i-th victim with a probability $\frac{p_i}{\sum_{j=1}^{N} p_j}$ (N : # of workers) for a single non-uniformly random forced steal attempt
- Advantage : act as PG without repeated steal attempts
- Disadvantage : refer to the probability of all victims



Outline

Contributions

Background

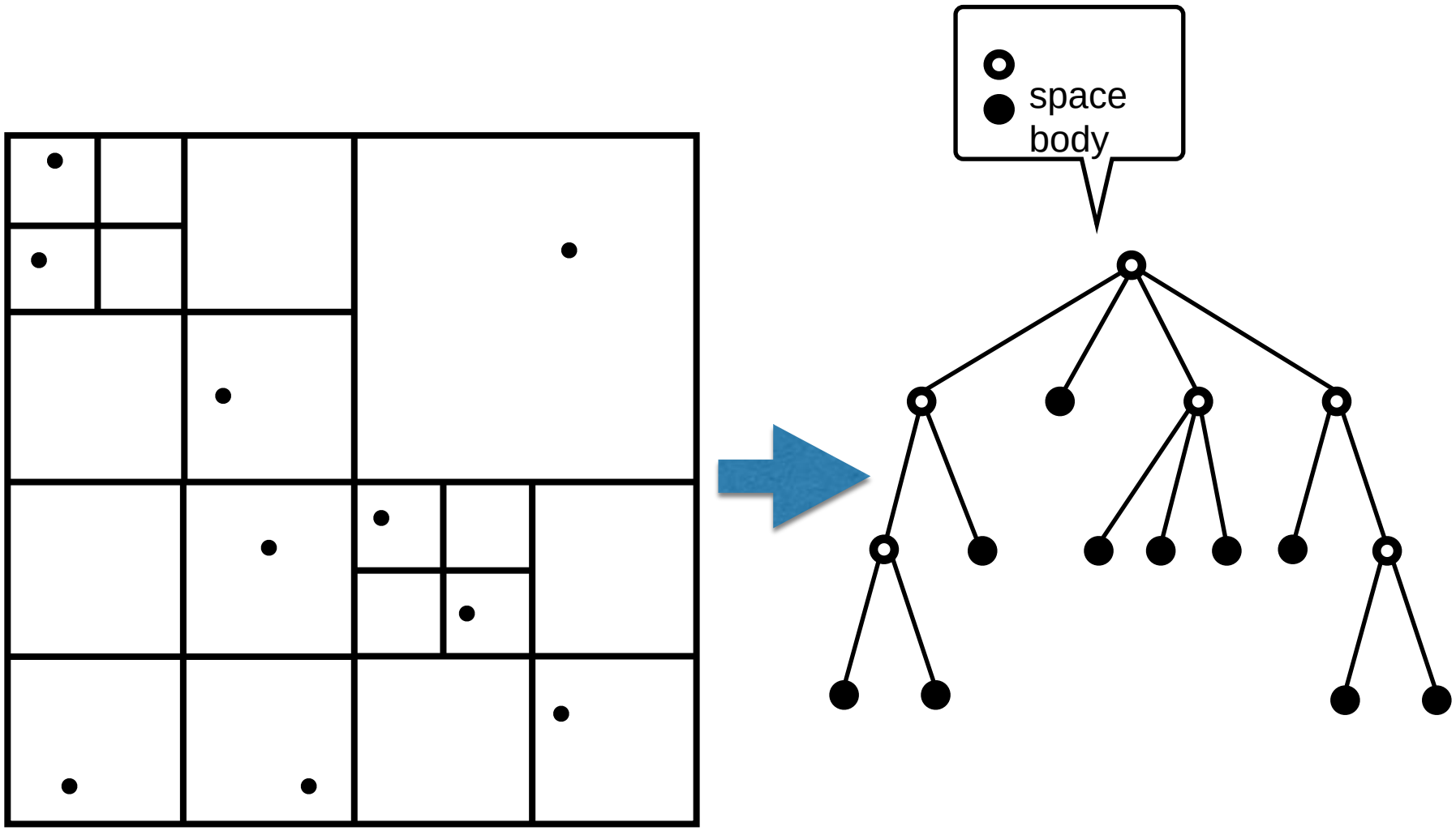
Tascell

Our proposals

Probabilistic guards
Virtual probabilistic guards

Evaluations

Barnes-Hut algorithm
Experimental results


space
body

Barnes-Hut algorithm

Barnes-Hut algorithm performs as follows
(repeats 1 to 3) in each time step

1. Constructs a tree structure representing space

2. Calculates forces for all bodies

3. Based on the result of the force calculations, updates the velocities and positions of all bodies

Outline

Contributions

Background

Tascell

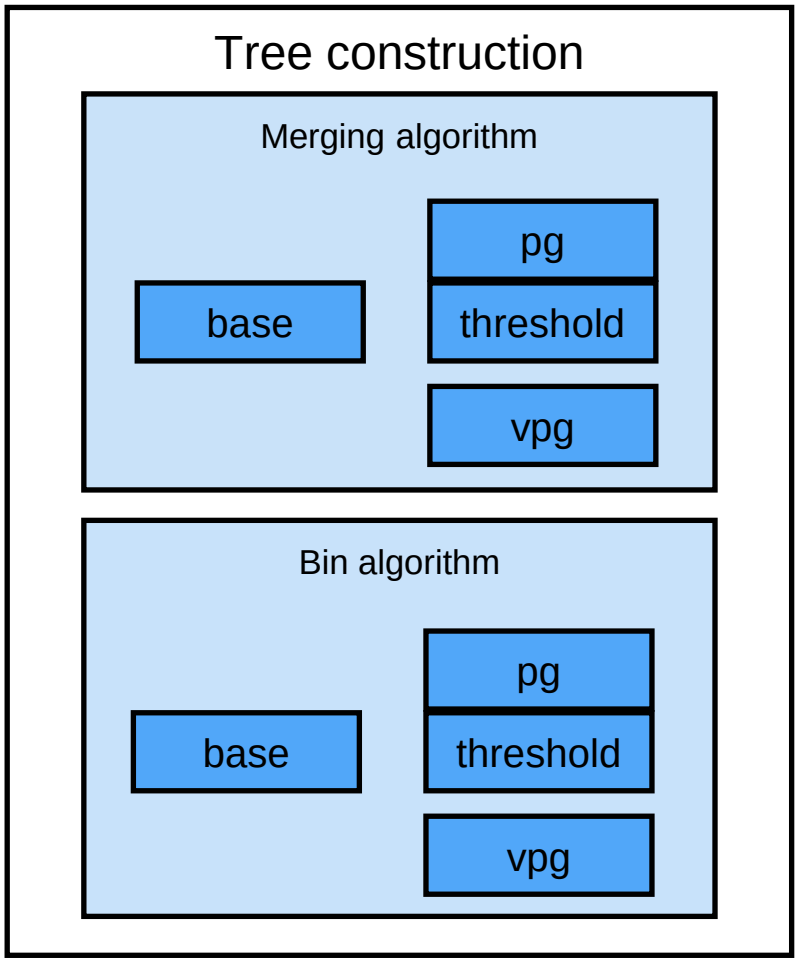
Our proposals

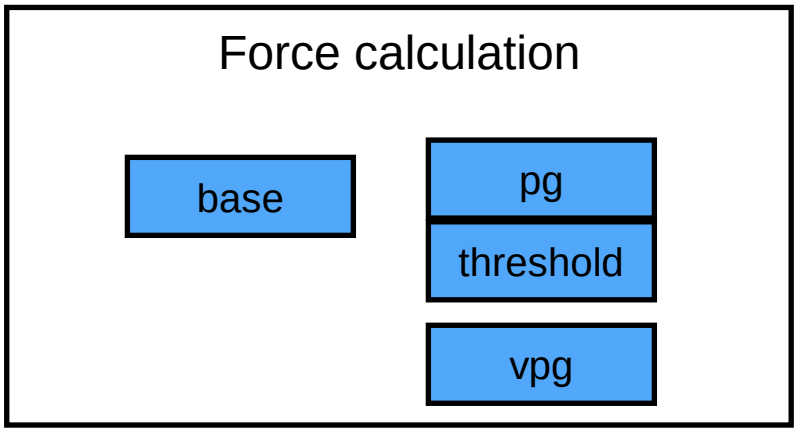
Probabilistic guards
Virtual probabilistic guards

Evaluations

Barnes-Hut algorithm
Experimental results

Programs for evaluations



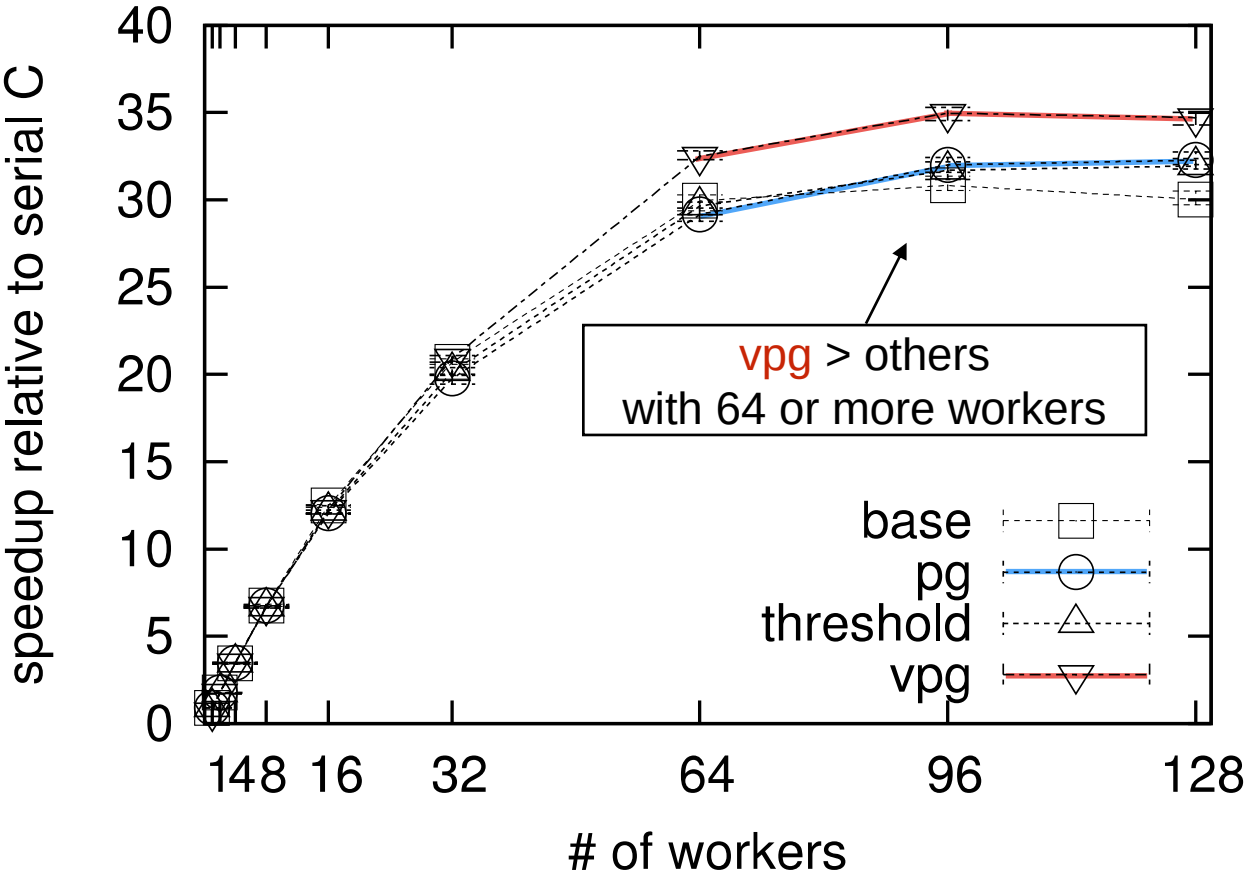


- upper limit : no limit
- base : Tascell without probabilistic guards
- pg : probabilistic guards
- threshold : probabilistic guards with thresholds
- vpg : virtual probabilistic guards

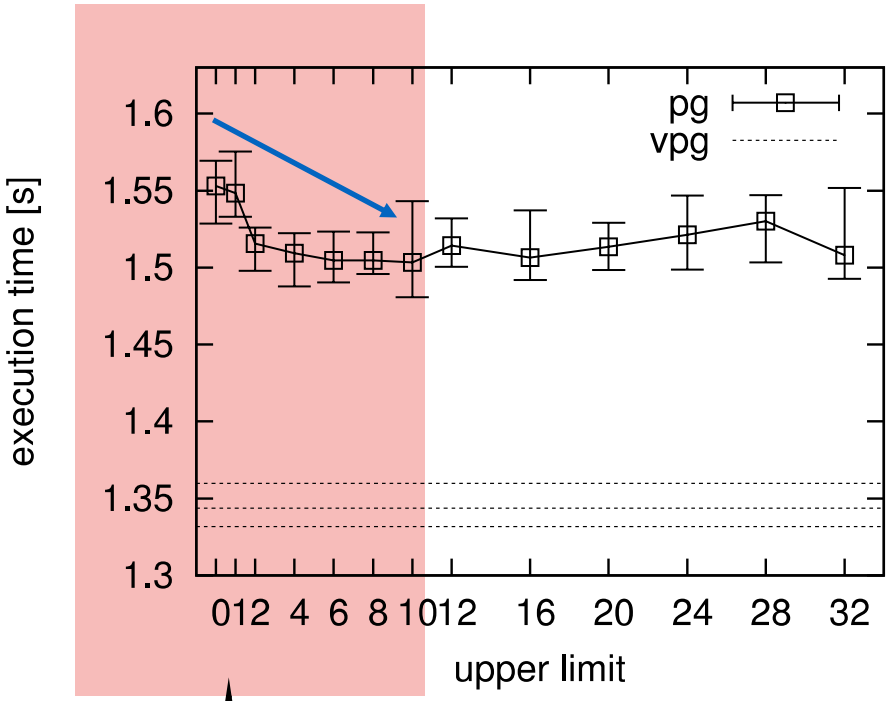
Evaluation environment

	Xeon Phi
(Co-)processor	Intel Xeon Phi 3120P 57-core
Memory	6GB
Compiler	Intel Compiler 13.1.3 with -02 optimizer

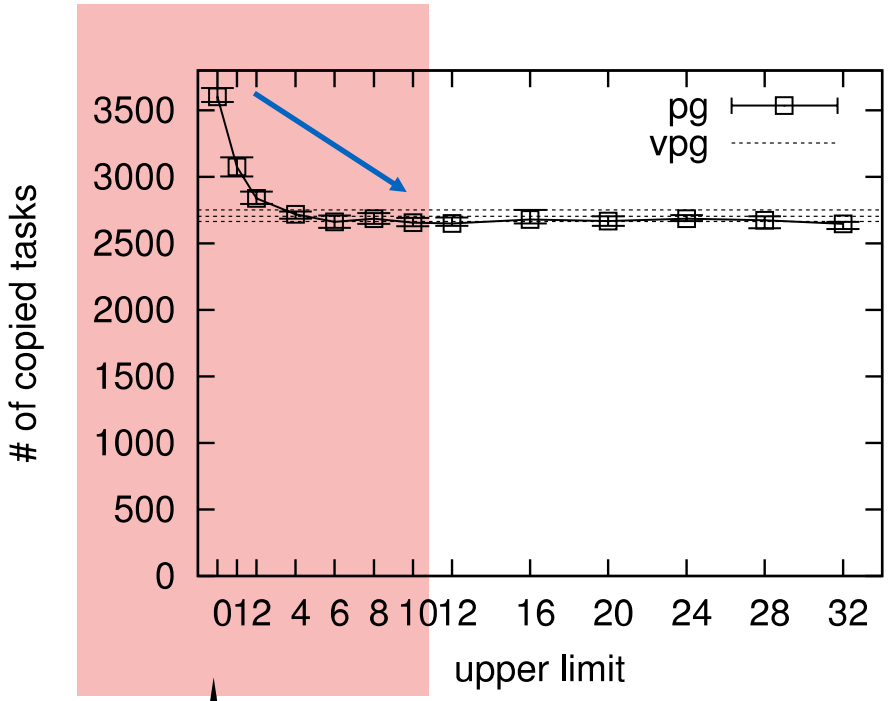
Performance evaluation / force calculation (500,000 bodies)



Effect of upper limit / force calculation (128 workers, 500,000 bodies)



Performance improvement



Decreasing the number of task division

Summary of evaluations

- Without upper limit:
  - For tree construction, the difference in performance is slight
    base = threshold = pg = vpg
    → the task division cost is relatively low
  - For force calculation,
    base < threshold = pg < vpg
    → PG and VPG is more effective because the task division cost is higher (“interaction list” is relatively large data)
    → Victim selection with VPG is quicker than PG

- When the upper limit is valid:
  - The performance of PG improves as the upper limit gets close to 10

発表内容

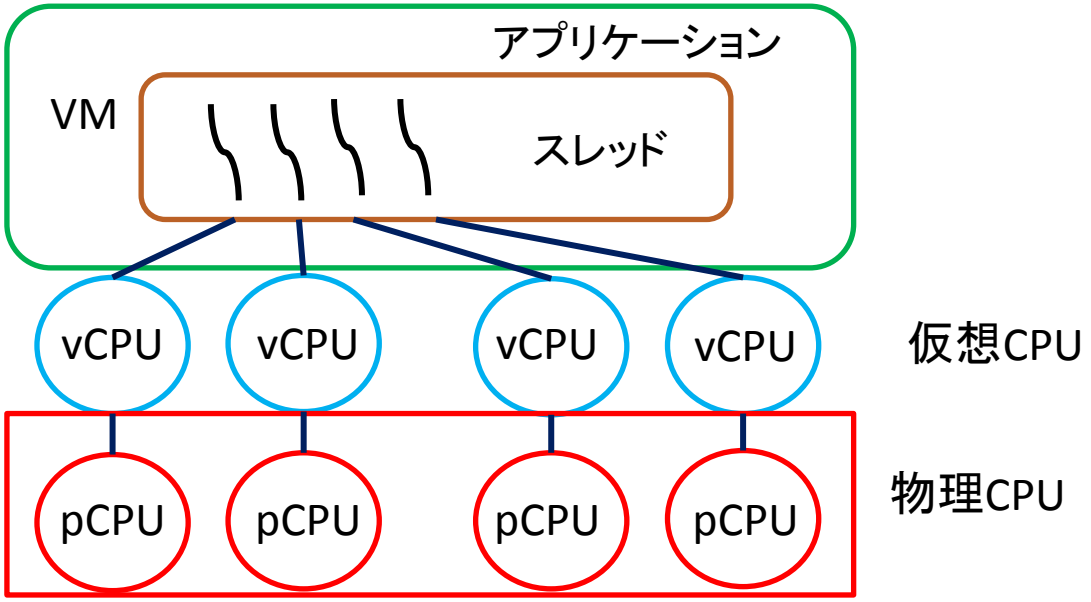
- (安全な)計算状態操作機構L-closureとは?
  - AT との関係
- これまでの応用例
- これまでの実装
- 最近の応用例
  - Tascell: 分散メモリ環境における共通アイテム集合を持つ連結部分グラフ抽出の並列化 [IPSJ-PRO研究会 2016]
  - Tascell: MPIベース実装，大規模並列環境での評価 [P2S2 2016]
  - Tascell: 確率的ガード [P2S2 2016]
  - 仮想環境における資源不足時のTascellの性能低下の確認 [JSSST大会 2016]
- 最近の実装
- これからの課題

VMが利用可能なCPU数の変化に対応した並列アプリケーション実行の最適化

九州工業大学
髙山 都旬子
光来 健一

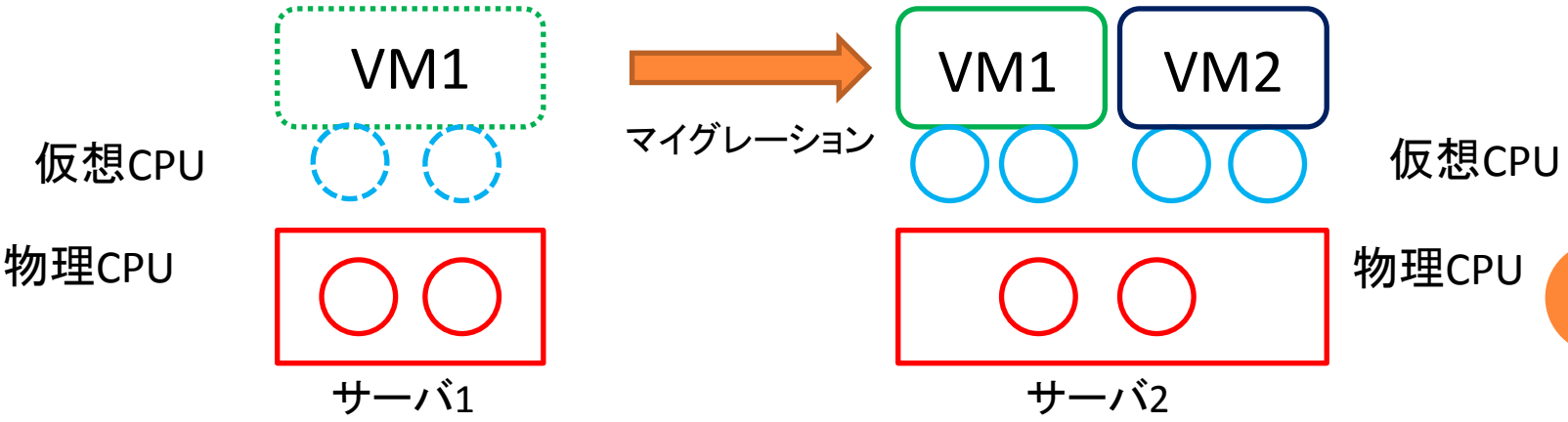
VM内での並列アプリケーションの実行

- 仮想マシン（VM）内で並列アプリケーションを動かすことも増えてきた
  - アプリケーションスレッドは仮想CPUに割り当てられる
  - 仮想CPUは物理CPU（CPUコア）に割り当てられる
    - 仮想CPUアフィニティを設定すると割り当てを固定



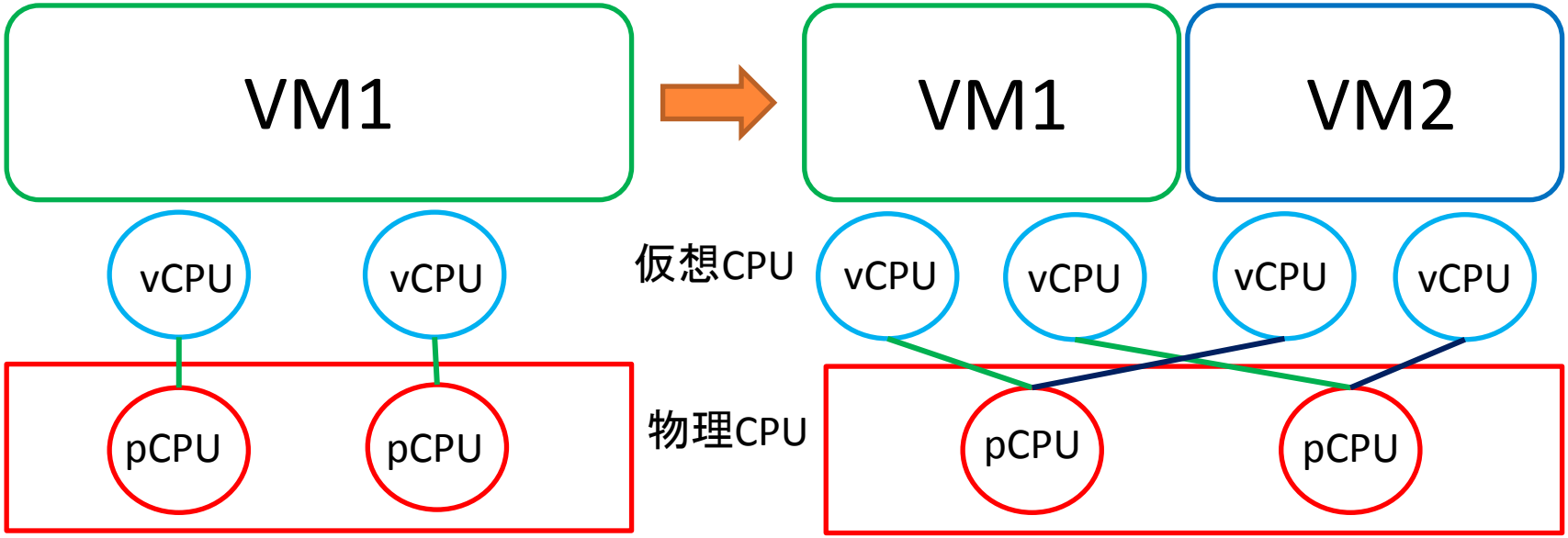
VMマイグレーションの影響

- VMは他のホストに移動されることがある
  - 例：サーバのメンテナンス、負荷分散
- マイグレーション先で物理CPUが不足する可能性
  - 物理CPUより多くの仮想CPUを動作させるのが一般的
    - オーバサブスクリプション
  - 並列アプリケーションはCPUを占有することが多い



物理 CPU の不足時の対処 (1/3)

- 複数のVM間で物理CPUを共有
  - 1つの物理CPUを複数のVMの仮想CPUに割り当てる
  - 仮想CPUが使える物理CPUの割合はVMの重みで決まる
    - 例：50%と50%（デフォルト）、33%と67%
    - 使い切らない場合は他の仮想CPUが使う

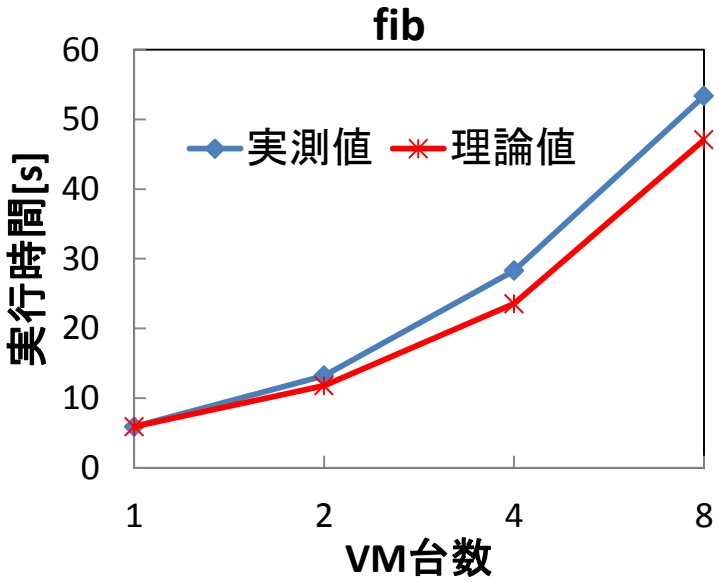


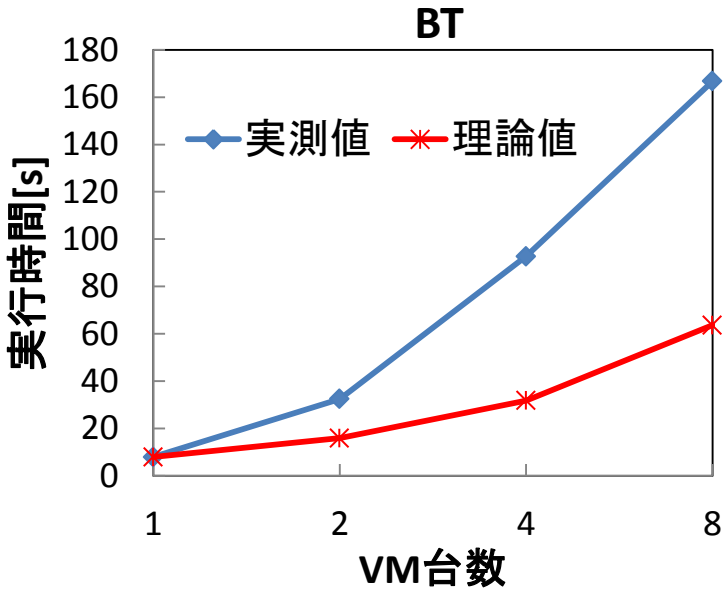
予備実験

- 並列アプリケーションの性能低下の調査
  - 物理CPUの不足時の三つの対処について
- 対象とした並列アプリケーション
  - Tascell [Hiraichi et al.'09] で並列化したフィボナッチ数計算
  - NAS Parallel Benchmarks (BT)
- 実験環境
  - CPU: AMD Operation 6367 (16コア) × 2，メモリ: 320GB
    - 物理CPUとして16コアのみを使用
  - 仮想化ソフトウェア: Xen 4.4.0
  - VM: 仮想CPU 16個，メモリ 4GB，Linux 3.13.0

複数のVM間で物理CPUを共有した場合

- VMの台数を1台から8台まで増加させた
  - 台数を増やすほど理論値より遅くなった
    - n台の時の理論値：1台の時の実行時間のn倍
  - 理論値からの増加率
    - fib: 12% ～ 20%, BT: 104% ～ 191%



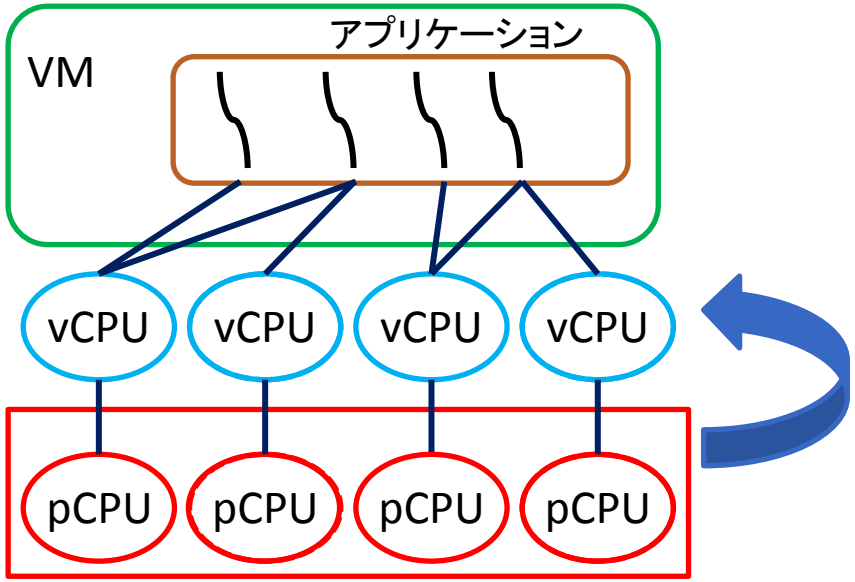


提案：pCPU-Est

- 物理CPUの不足時に並列アプリケーションの実行を最適化
  - 仮想CPUの切り替えを減らすための最適化を行う
    - VM の仮想CPU 数の最適化
    - アプリケーションスレッド数の最適化
  - そのために、VMが実際に利用可能な物理CPU数を見積もる

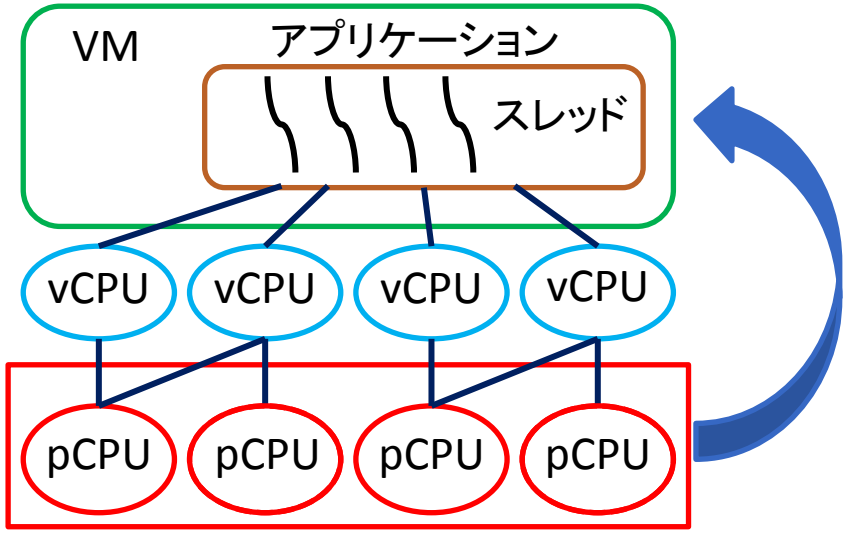
VMの仮想 CPU 数の最適化

- 仮想CPUを減らすことでスケジューリングを回避
  - VMが利用可能な物理CPU数に応じて調整
    - 並列度をなるべく保ちつつ、仮想CPUの切り替えを減らす
- 仮想化システムの管理者のみ変更可能
  - システム全体に影響が及ぶ可能性も



アプリケーションスレッド数の最適化

- スレッドを減らすことで仮想CPUを間接的に減らす
  - 使われない仮想CPUはスケジュールされないことを利用
  - VMが利用可能な物理CPU数に応じて調整
- ユーザが容易に調整できる
  - 実行中に変更可能かどうかはアプリケーション依存

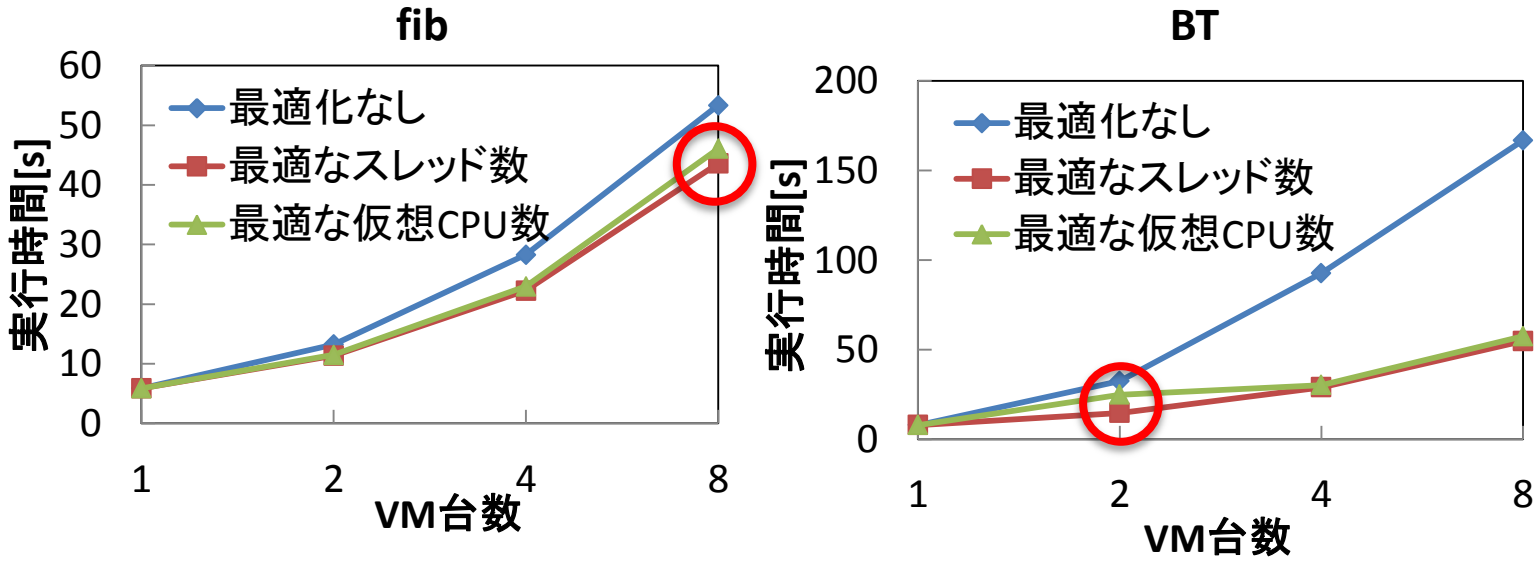


実験

- 目的
  - 最適なアプリケーションスレッド数・仮想CPU数を調査
    - 物理CPU不足時の三つの対処それぞれについて
  - pCPU-Estによる最適化の効果を確認
    - 最適なスレッド数・仮想CPU数を用いた
- 対象とした並列アプリケーション
  - 並列フィボナッチ数計算 (fib)
  - NAS Parallel Benchmarks (BT)
- 実験環境
  - 予備実験と同じ

最適化の効果：物理CPUの共有時

- どちらの最適化でも性能が改善された
  - 1.2～3.2倍の高速化
  - 特に理論値からの増加率が大きかったBTで改善
- スレッド数の最適化のほうがよい場合があった
  - VMが8台の時のfib (6%)、VMが2台の時のBT (69%)



まとめ

- 物理CPUの不足時の対処を行った際の並列アプリケーションの性能への影響を調査
  - 利用可能な物理CPUの減少分以上に性能が低下
- 並列アプリケーションを最適に実行するpCPU-Estを提案
  - VMが利用可能な物理CPU数を見積もる
  - アプリケーションスレッド数・仮想CPU数を最適化
  - スレッド数の最適化がより効果的

発表内容

- (安全な)計算状態操作機構L-closureとは?
  - AT との関係
- これまでの応用例
- これまでの実装
- 最近の応用例
- 最近の実装 (→ 新展開に向けて)
- これからの課題

発表内容

- (安全な)計算状態操作機構L-closureとは?
  - AT との関係
- これまでの応用例
- これまでの実装
- 最近の応用例
- 最近の実装
- これからの課題
  - 新展開に向けて

新展開に向けて: 応用

- HOPE並列実行モデル (ACSI 2015にて頭出し)
- LW-SC上のSchemeインタプリタ (開発中)
- Tascell の VM 上での動作への対応
  - マイグレーションなどで仮想CPU数と比較して，物理CPU(コア)数が少なくなる場合，時分割による仮想CPUに
    - polling が遅れる問題
    - 解決のための参考技術
      - Indolent Task Creation [Strumpen 1998] (MIT Cilkの変種)
      - 仮想環境にスケジューリングを依頼
  - 現状: 勝手にスティール可能なタスクを持たせたTascellの新しい実装が完了．評価中

新展開に向けて:設計・実装

- 例外処理を用いたコードへの変換による実装
  - これまでの変換による実装では return が，本当のリターンなのか L-closure 呼び出し等のためなのかに確認するコードが必要だった
  - 例外処理を利用して，できれば軽量に
- Java言語における L-closure
  - 拡張Java言語 から 標準Java言語への変換で実装
    - 例外に，消されそうなFrame情報を含め，あとで再開可能に
- ベースとして：Java言語に関する変換に基づく拡張を容易とする処理系を準備中
  - 新しいリスト接合パターンの導入

本年(度)の発表(1/2)

- 重本 孝太, 八杉 昌宏, 平石 拓, 馬谷 誠二. HOPEコンパイラの実装に向けて. 日本ソフトウェア科学会プログラミング論研究会第18回プログラミングおよびプログラミング言語ワークショップ (PPL2016) (カテゴリ3, ポスター), 岡山県玉野市, March 2016.
- 良本 海, 八杉 昌宏, 平石 拓, 馬谷 誠二. 仮想環境を考慮した要求駆動型負荷分散の検討. 日本ソフトウェア科学会プログラミング論研究会第18回プログラミングおよびプログラミング言語ワークショップ (PPL2016) (カテゴリ3, ポスター), 岡山県玉野市, March 2016.
- Tasuku Hiraishi, Shingo Okuno, and Masahiro Yasugi. An Implementation of Exception Handling with Collateral Task Abortion. Journal of Information Processing, Vol. 24, No. 2, pp. 439-449, March 2016.
- Shingo Okuno, Tasuku Hiraishi, Hiroshi Nakashima, Masahiro Yasugi, and Jun Sese. Reducing Redundant Search using Exception Handling in a Task-Parallel Language. 21st International Workshop on High-Level Parallel Programming Models and Supportive Environments HIPS 2016 (held in conjunction with IPDPS 2016) (Proc. of IPDPSW 2016, pp. 328-337), Chicago, May 23rd, 2016.
- 奥野 伸吾, 平石 拓, 中島 浩, 八杉 昌宏, 瀬々 潤. 分散メモリ環境における共通アイテム集合を持つ連結部分グラフ抽出の並列化. 情報処理学会第109回プログラミング研究会, 浜松市, June 2016.
- Tasuku Hiraishi, Shingo Okuno, Daisuke Muraoka, and Masahiro Yasugi. Exception Handling with Collateral Task Abortion in Distributed Memory Environments. ISC 2016 HPC in Asia Posters, Frankfurt, Germany, June 2016.

本年(度)の発表(予定)(2/2)

- Daisuke Muraoka, Masahiro Yasugi, Tasuku Hiraishi, and Seiji Umatani. Evaluation of an MPI-Based Implementation of the Tascell Task-Parallel Language on Massively Parallel Systems. Ninth International Workshop on Parallel Programming Models and Systems Software for High-End Computing P2S2 2016 (held in conjunction with ICPP 2016), (Proc. of ICPPW 2016, pp. 161-170), Philadelphia, August 16th, 2016.
- Hiroshi Yoritaka, Ken Matsui, Masahiro Yasugi, Tasuku Hiraishi, and Seiji Umatani. Extending a Work-Stealing Framework with Probabilistic Guards. Ninth International Workshop on Parallel Programming Models and Systems Software for High-End Computing P2S2 2016 (held in conjunction with ICPP 2016), (Proc. of ICPPW 2016, pp. 171-180), Philadelphia, August 16th, 2016.
- 髙山都旬子, 光来健一. VMが利用可能なCPU数の変化に対応した並列アプリケーション実行の最適化. 日本ソフトウェア科学会第33回大会, 2016年9月8日.
- 江本 健斗, 松崎 公紀, 胡 振江, 森畑 明昌, 岩崎 英哉. 大規模グラフ並列処理のための関数型領域特化言語 Fregel とその評価. 日本ソフトウェア科学会第33回大会, 仙台市, 2016年9月7日-9日.
- Kento Emoto, Kiminori Matsuzaki, Zhenjiang Hu, Akimasa Morihata, and Hideya Iwasaki. Think like a vertex, behave like a function! a functional DSL for vertex-centric big graph processing. In Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, ICFP2016, Nara, Japan, Setember 19-21, 2016, pp. 200-213, ACM, 2016.
- Shingo Okuno, Tasuku Hiraishi, Hiroshi Nakashima, Masahiro Yasugi, and Jun Sese. Parallelization of Extracting Connected Subgraphs with Common Itemsets in Distributed Memory Environments. Journal of Information Processing, 2016. (To appear).

ご清聴ありがとうございます