安全な計算状態操作機構の実用化

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）
(基盤研究(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
- 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

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

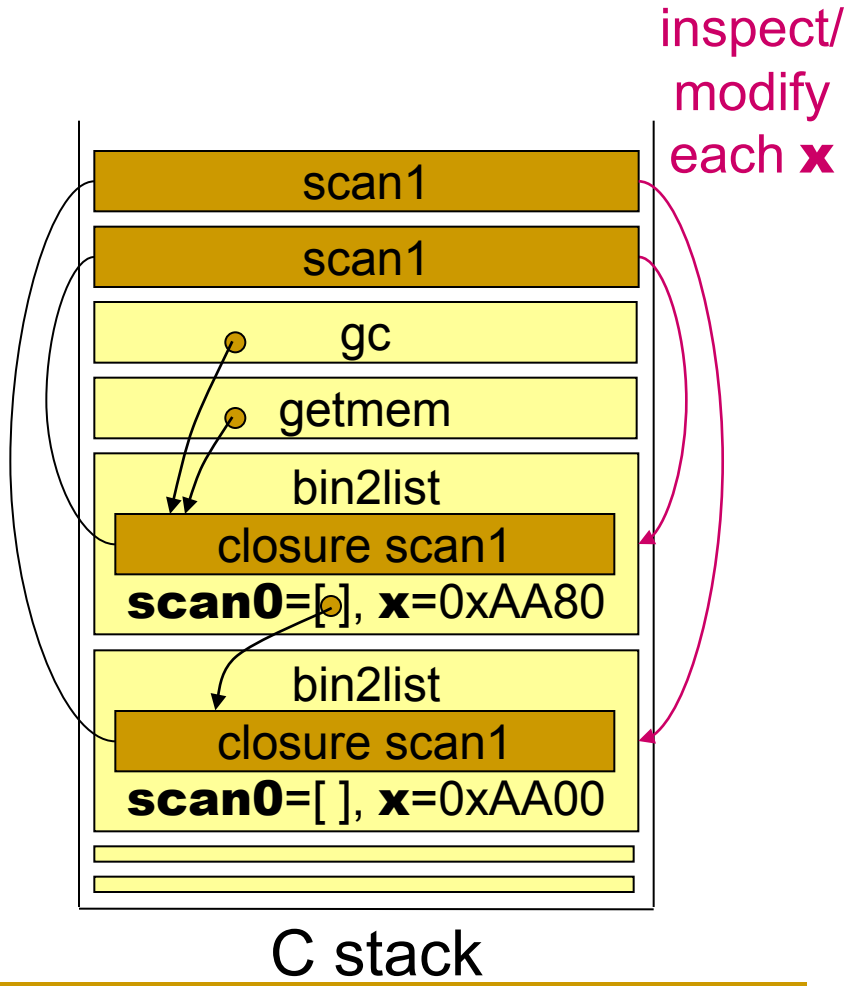
L-closure: 計算状態操作機構とは? (2/2)

- 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]

```
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)

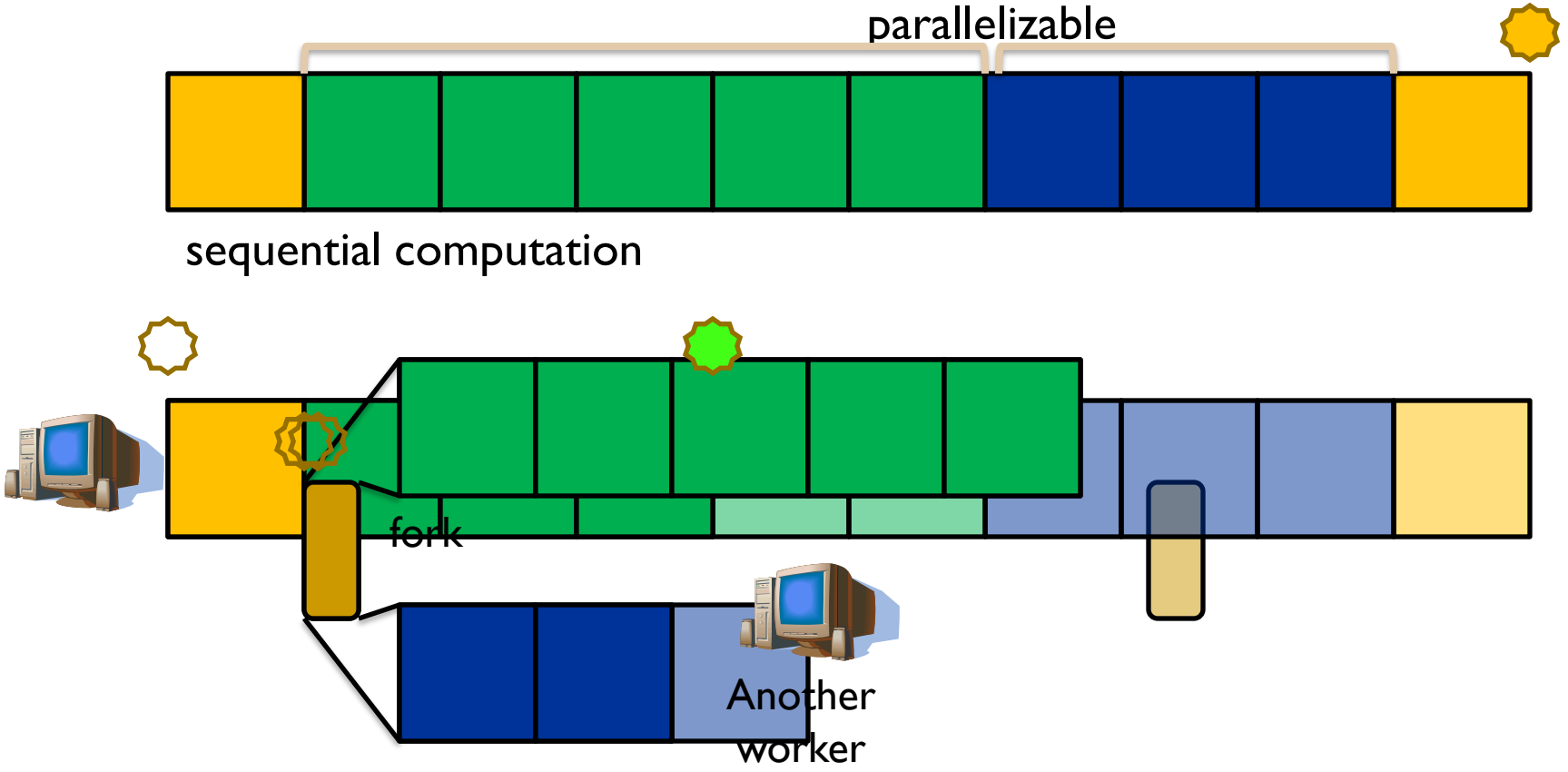
- ある種のジレンマ
  - あるrunで，常に同じ選択ではいけない
- ゲーム理論の混合戦略(ある確率で選択)は?
  - 選択すべき時にはまだ情報不足
- 後から選択を変更できるとよいが?
  - 計算状態操作機構で可能
    (呼び出し元の変数の値の参照・変更などにより)
- 負荷分散の例
  - 逐次実行を後から並列実行に変更

発表内容

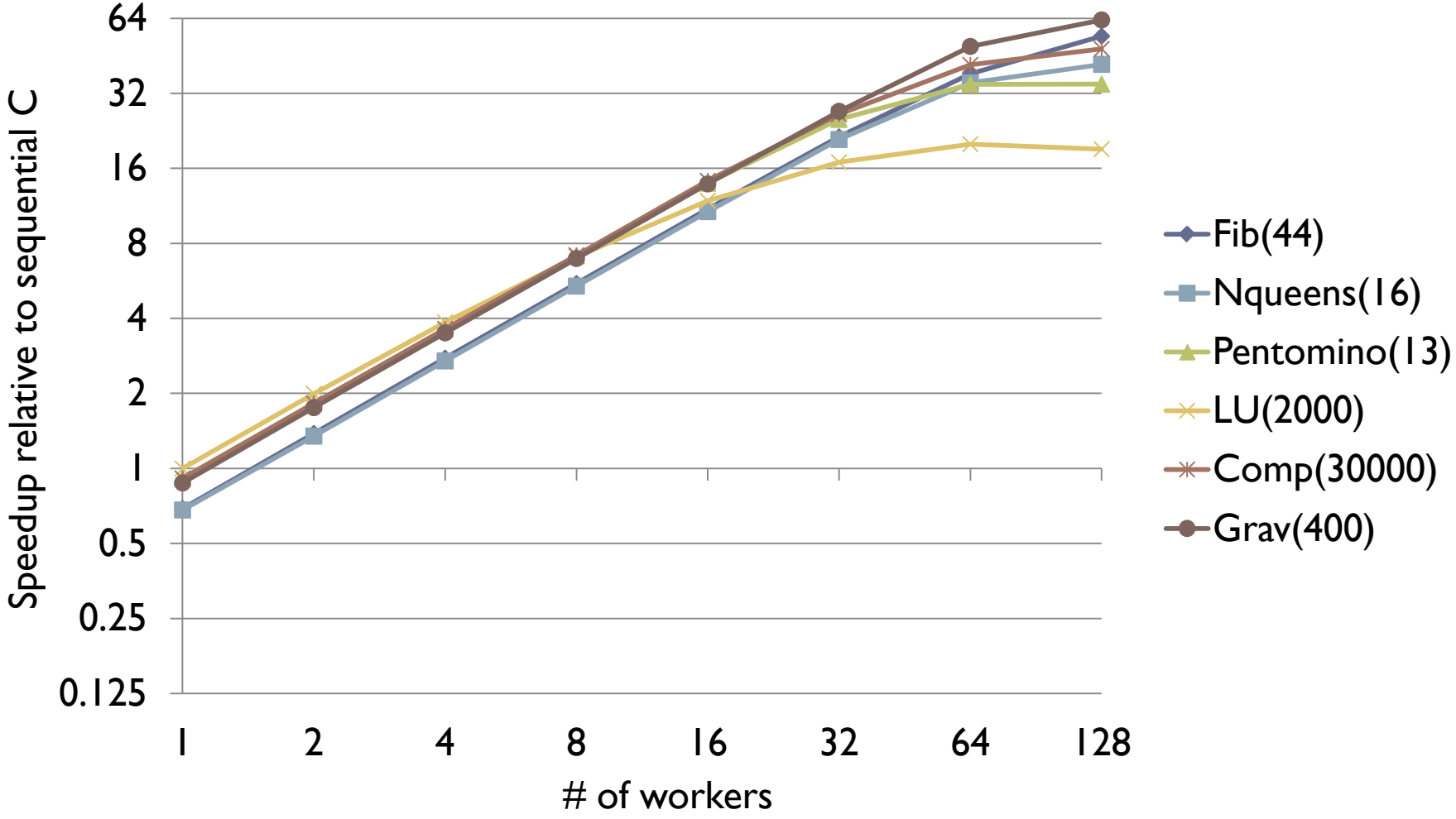
- (安全な)計算状態操作機構L-closureとは?
  - AT との関係
- これまでの応用例
  - Tascell: Backtracking-based Load Balancing [平石・八杉 他 PPoPP 2009]
  - 真の末尾再帰 (proper tail recursion) [ILC 2010]
  - Tascell: spanning tree construction [ICPADS 2010]
  - Tascell: 広域分散環境への適用 [SACSIS 2011]
- これまでの実装
- 最近の応用例
- 最近の実装
- これからの課題

Tascell: Load Balancing Framework [PPoPP2009]

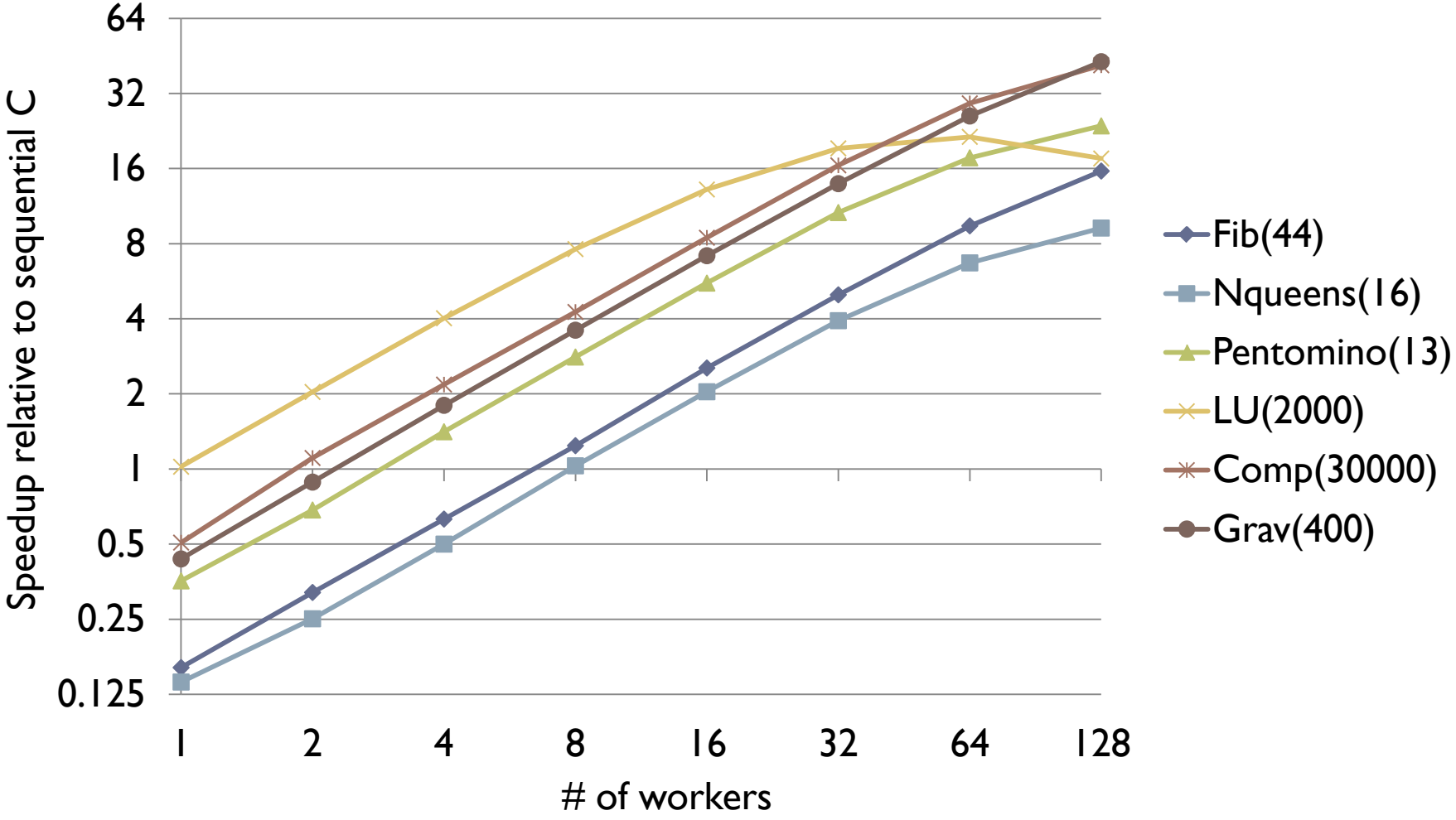
- *"Logical thread"-free*
- Compute sequentially, fork only when requested



測定結果（Niagara2 Tascell）



測定結果（Niagara2 Cilk [PLDI'98]）

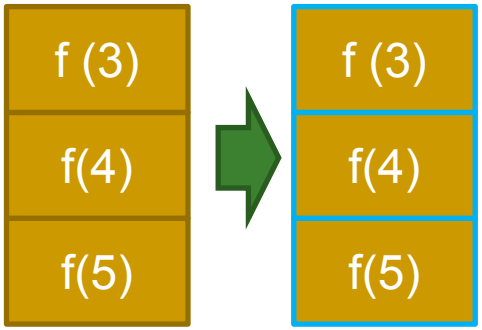


発表内容

- (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
  - 標準C言語への翻訳
  - GCC拡張の実装モデル
    - GCC 3.4.6ベース, SPARC, IA-32
- 最近の応用例
- 最近の実装
- これからの課題

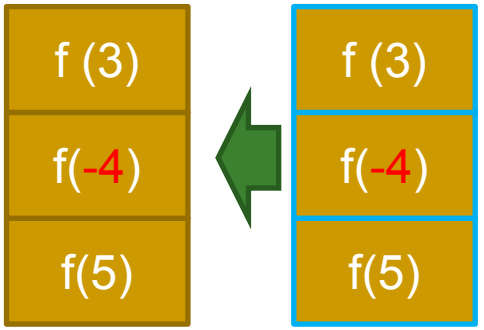
Transformation-based implementation of L-closures [Hiraishi,Yasugi,Yuasa2005,6]

- Translating into “standard” C
- Using an explicit stack
  - Save C frames as explicit frames and “return”
  - Call the L-closure which inspects/modifies data on the explicit frames



Transformation-based implementation of L-closures [Hiraishi,Yasugi,Yuasa2005,6]

- Translating into “standard” C
- Using an explicit stack
  - Save C frames as explicit frames and “return”
  - Call the L-closure which inspects/modifies data on the explicit frames
  - “Call” and restore C frames from the *modified* explicit frames



GCC-based Implementation Model for L-Closures

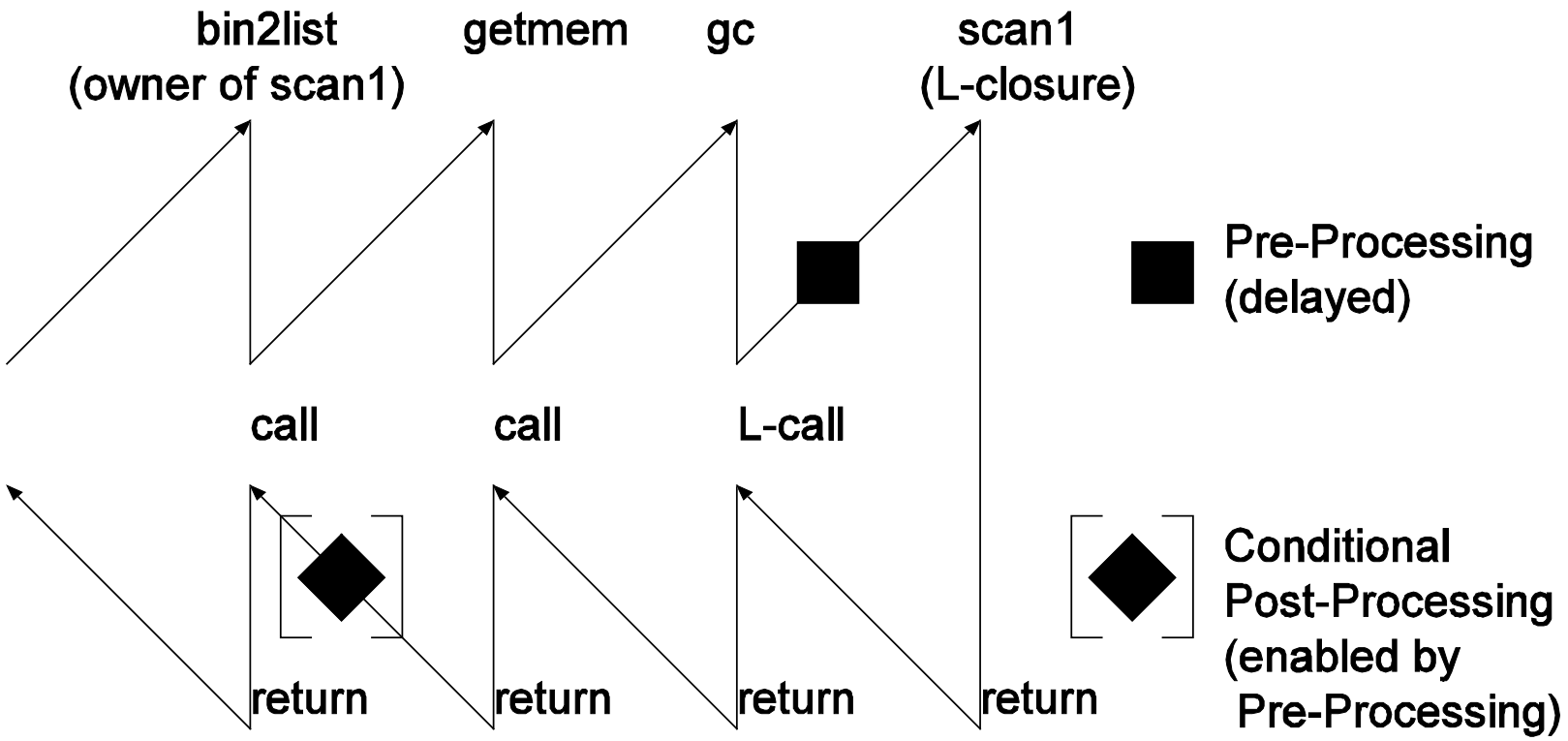
- Two locations for each variable
  - *Private* location
    - Compiler's pseudo register
    - Private values may be kept in registers.
  - *Shared* location
    - Stack frame slot
    - Accessed by L-closure bodies.
- The coherency between private and shared locations is maintained *lazily*.

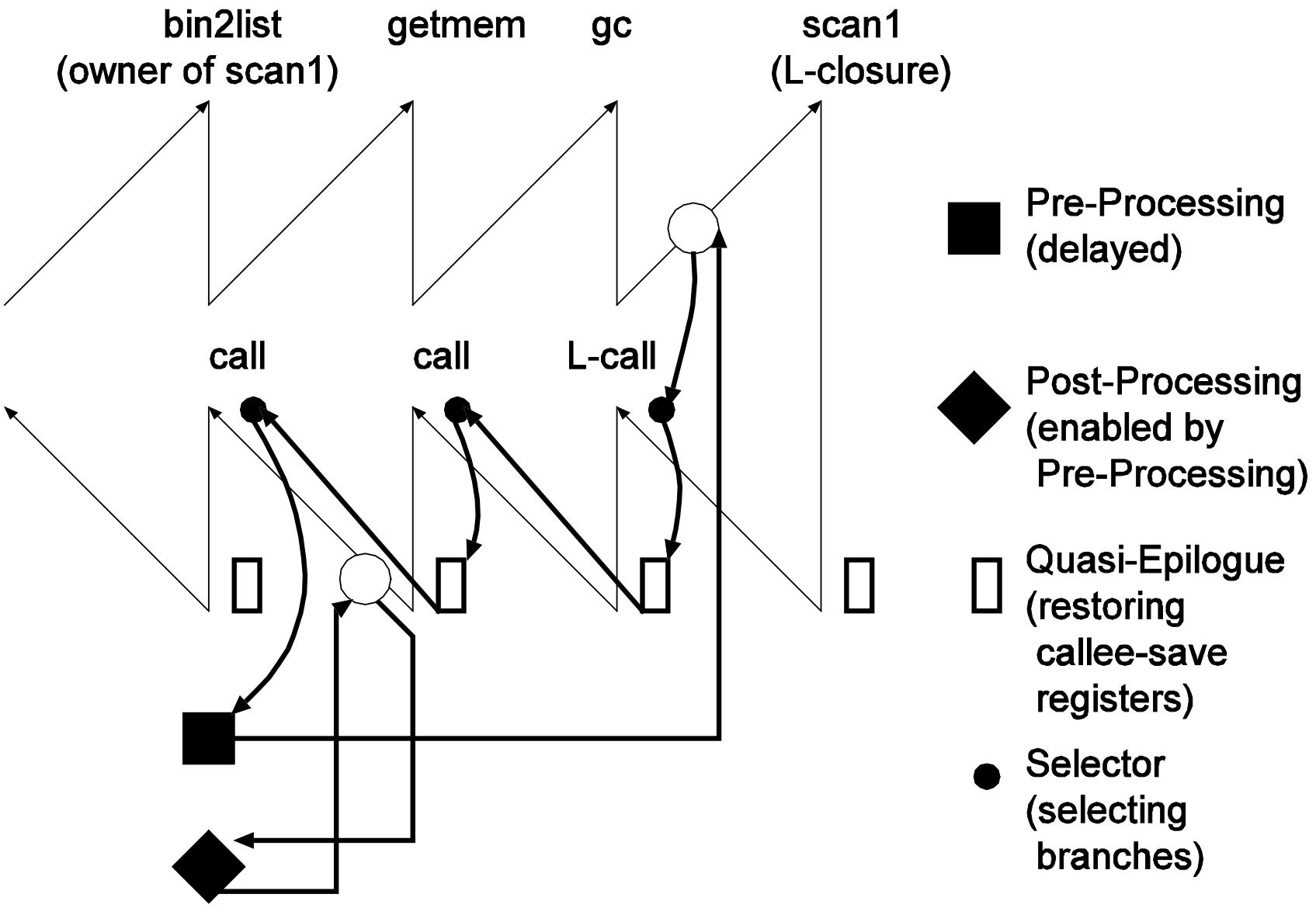
Implementation Model for L-Closures

- For efficiency, Pre-processing should be *delayed* until the L-closure is *actually invoked*.
- Delayed (lazy) pre-processing
  - Save private values ➔ shared locations
  - Initialize the code-environment pair
  - Enable post-processing
- Conditional post-processing
  - Restore private values ➔ shared locations

Lazy Pre-/Post-Processing for L-Closures

- Delayed until the L-closure is actually invoked



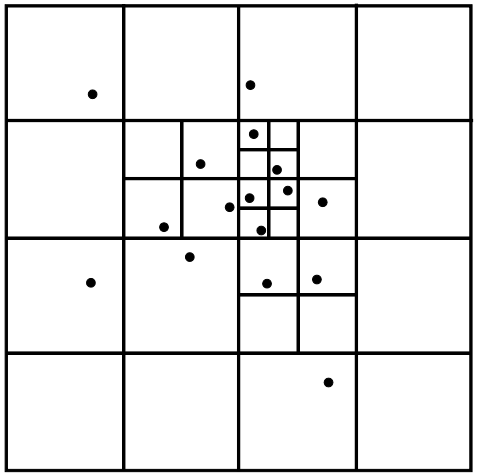

bin2list
(owner of scan1)
getmem
gc
scan1
(L-closure)
call
call
L-call
Pre-Processing
(delayed)
Post-Processing
(enabled by
Pre-Processing)
Quasi-Epilogue
(restoring
callee-save
registers)
Selector
(selecting
branches)

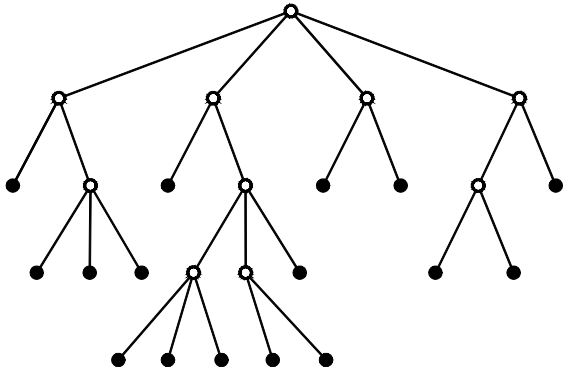
発表内容

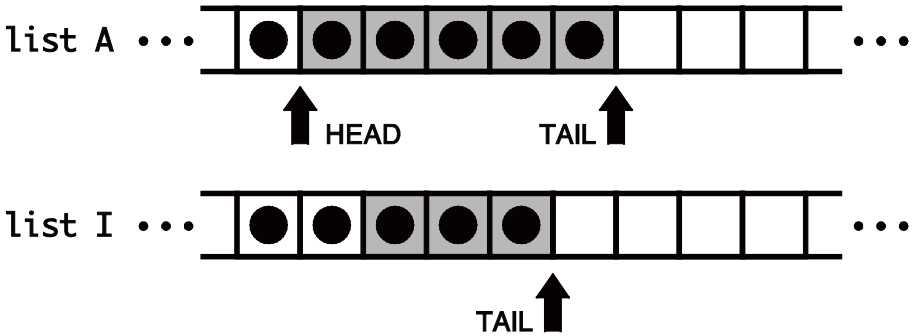
- (安全な)計算状態操作機構L-closureとは?
  - AT との関係
- これまでの応用例
- これまでの実装
- 最近の応用例
  - Tascell: *N*体問題（Barnes-Hut algorithm）[SACSIS 2012]
- 最近の実装
- これからの課題

多体問題(Barnes-Hut algorithm)

- 高速版Barnes-Hut 多体シミュレーションの並列実装







性能評価(共有メモリ環境)

- コンパイラ: ここでは単にGCC 4を利用

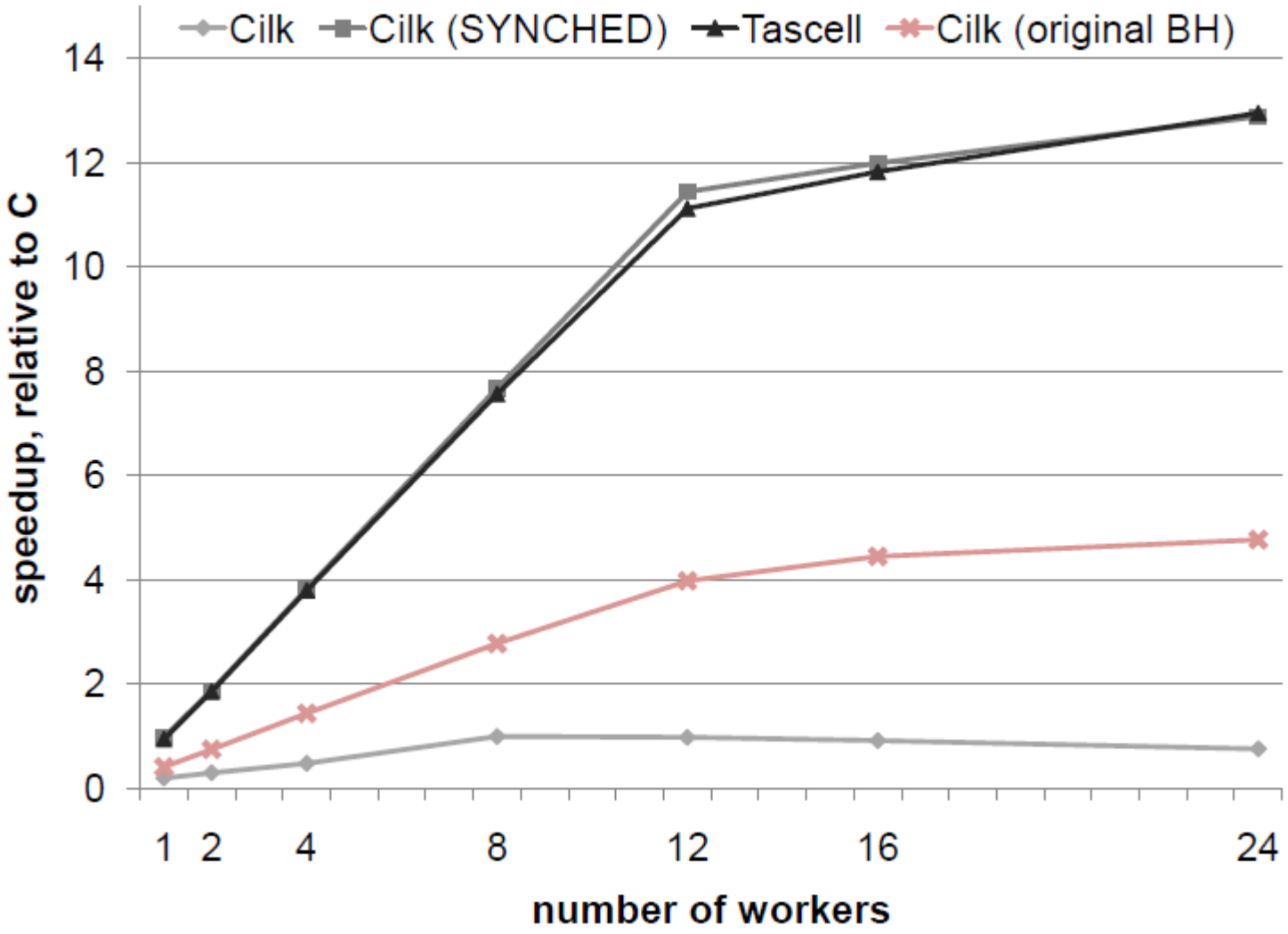
	Linux PC Server (x86-64)
CPU	Xeon X5650 2.67GHz 6-Core×2 Hyper-threading を有効化 (計 24 スレッド) Turbo Boost を無効化
メモリ	24GB
OS	CentOS 5.4 (64bit)
コンパイラ	Tascell コンパイラ + GCC 4.1.2 Cilk コンパイラ 5.4.6 + GCC 4.4.2 最適化オプション -O3 を有効化

性能評価(共有メモリ環境)

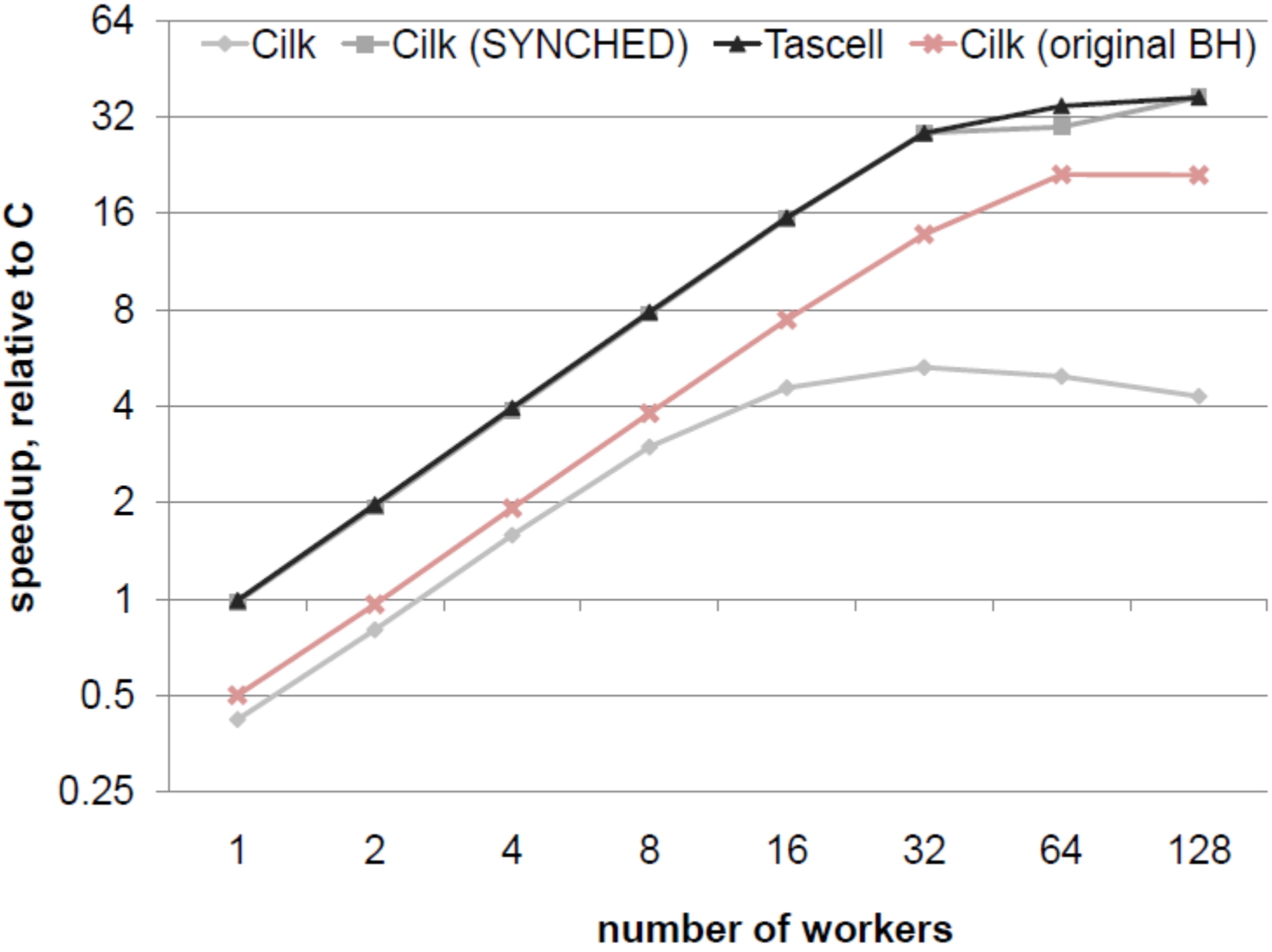
- コンパイラ: ここでは単にGCC 4を利用

	UltraSPARC T2 Plus Server (SPARC)
CPU	UltraSPARC T2 Plus 1.4GHz 8-Core×2 コア当たり 8 スレッド (計 128 スレッド)
メモリ	24GB
OS	SunOS 5.10
コンパイラ	Tascell コンパイラ + GCC 4.4.2 Cilk コンパイラ 5.4.6 + GCC 4.4.2 最適化オプション -O3 を有効化

評価結果 (力の計算: x86-64)



評価結果 (力の計算:SPARC)



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
  - 標準C言語への翻訳
  - GCC拡張の実装モデル
    - GCC 3.4.6, SPARC, IA-32
- 最近の応用例
- 最近の実装
  - x86-64 のサポート, callee-save レジスタ利用増加は逆効果も
  - GCC バージョン4 ベースへ, ただし closure の実装
  - 償却時間(amortized time)を下げる実装モデル for 翻訳による実装
- これからの課題

GCC 4 ベースの新しい closure 実装

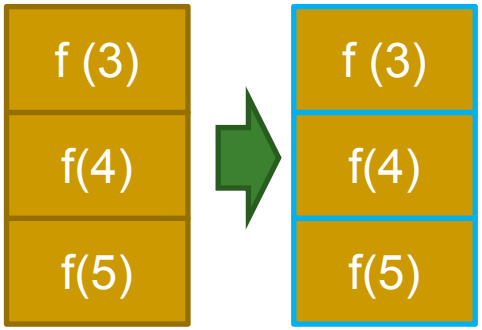
- Closure
  - L-closureよりも呼び出しコスト低い
  - ただし，生成コスト，維持コストが必要
- GCC 4.6.3 を拡張
  - GCC 3.4.6 との違い
    - RTL 以外に，GENERIC, GIMPLE 中間言語(中間表現)
    - トランポリンの実装で，GIMPLE上の組み込み関数で表現
    - 外側の変数は構造体のフィールドに変換
    - GIMPLE 上で SSA 最適化等

翻訳に基づく新しい実装

- 償却時間(amortized time)を下げる実装モデル
- 翻訳を容易とする処理系(SC言語処理系)を利用
- 従来，翻訳による実装では，Cのスタックとは別のスタック(明示的スタック)を用意し，L-closure呼び出し時にCのスタックの内容を明示的スタックへ一時的に移すことで，入れ子関数を持つ関数の局所変数へのアクセスを実現していた.
- 新しい実装では，L-closureからのリターンの後，Cスタック全体を再構築するのではなく，フレーム毎に再構築することで，再度L-closureが呼び出された時のスタック間の値の移動を減らし，呼び出しコストの削減を実現した.

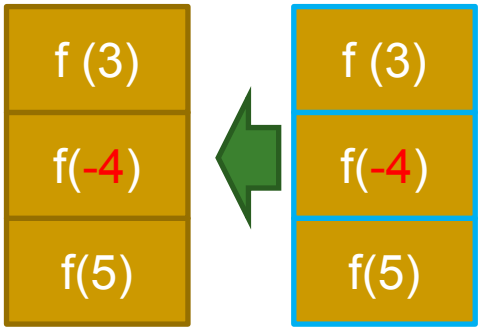
Transformation-based implementation of L-closures [Hiraishi,Yasugi,Yuasa2005,6]

- Translating into “standard” C
- Using an explicit stack
  - Save C frames as explicit frames and “return”
  - Call the L-closure which inspects/modifies data on the explicit frames



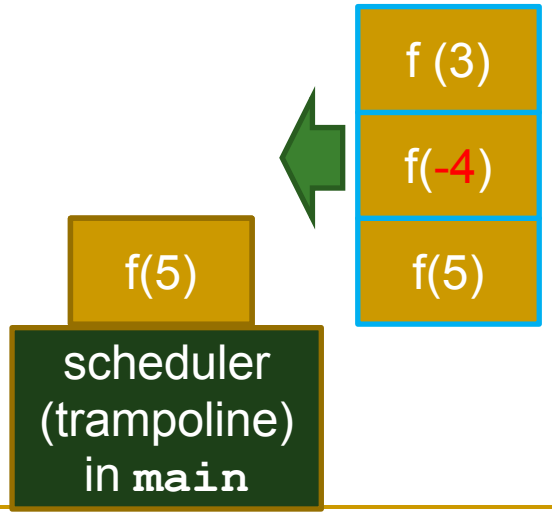
Transformation-based implementation of L-closures [Hiraishi,Yasugi,Yuasa2005,6]

- Translating into “standard” C
- Using an explicit stack
  - Save C frames as explicit frames and “return”
  - Call the L-closure which inspects/modifies data on the explicit frames
  - “Call” and restore C frames from the *modified* explicit frames



New Transformation-based implementation of L-closures

- Translating into “standard” C
- Using an explicit stack
  - Save C frames as explicit frames and “return”
  - Call the L-closure which inspects/modifies data on the explicit frames
  - “Schedule” and restore C frames from the *modified* explicit frames *one by one*



新しい翻訳に基づく実装の特徴

- L-closure 呼び出し中，呼び出し後も，明示的スタックにフレームは残る.
  - そのまま，Cスタック上で新たな呼び出し可能
- 再度L-closureが呼び出されたとき，以前保存したCフレームは保存する必要がない
- スケジューラ(トランポリン)は，任意のC関数が呼び出せないといけない
  - 直接は無理
  - 引数と結果を中継する(だけの)補助関数を自動生成

新しい実装らの性能

- 評価中
- GCC 4 ベースの closure実装
  - 多体問題やLU分解などで GCC 3 ベースより高速
  - 低い呼び出しコスト
  - 生成コスト・維持コストも悪くない
- 翻訳ベースの新しい実装
  - 翻訳後，最新のコンパイラが利用可能
  - GCC4拡張での L-closure 実装が困難なので…
  - SPARC では特に有効

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

これからの課題

- 短期的
  - 償却時間(amortized time)を下げる L-closure 実装
    モデルのGCC バージョン4 ベース実装 (できれば)
  - 他言語（e.g., Java）での実現
  - Tascellの拡張
- 長期的
  - 信頼性向上: 計算状態の保存と復元
  - 計算状態の保全と再利用に基づく
    新しい並列実行モデル

本年度(本年)の発表

- 松井健, 平石拓, 八杉昌宏, 馬谷誠二. 高速版 Barnes-Hut 多体シミュレーションの並列実装. 先進的計算基盤システムシンポジウム (SACSIS2012), pp. 298-306, May 2012.
- 松田友希, 八杉昌宏, 鵜川始陽. コードシェーカ: コード配置効果を考慮した適正な性能評価システム. 第14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論文集(PPL2012), pp. 115-129, March 2012.
- 松井健, 平石拓, 八杉昌宏, 馬谷誠二. ワークスティーリングフレームワークにおける集団通信機能. 情報処理学会研究報告 (SWoPP '12), Vol. 2012-HPC-135, No. 10, pp. 1-8, July 2012.