実際的なMMOデータベースシミュレーション

佐藤　剛宣
Derek Laufenberg
Robert Greene
Versant Corp.

今回取り組む課題

課題

マルチコアでスケールする
メモリ上の共有オブジェクト操作性能

例題

85,000 v.s. 80,000
同時プレイ
関ヶ原の合戦　MMO FPS

動機

As CPU cores become both faster and more numerous, **the limiting factor for most programs is now, and will be for some time, memory access**. Hardware designers have come up with ever more sophisticated memory handling and acceleration techniques–such as CPU caches–**but these cannot work optimally without some help from the programmer**.

Ulrich Drepper
"What every programmer should know about memory"

例えば

Tony Albrecht's Latency Elephant Article

Mike Acton's Sketches on Concurrency, Data Design and Performance

Data-Oriented Design on Game Developer Magazine

~ Trends for 2009 in Retrospect ~
aigamedev.com

例えばグラフで

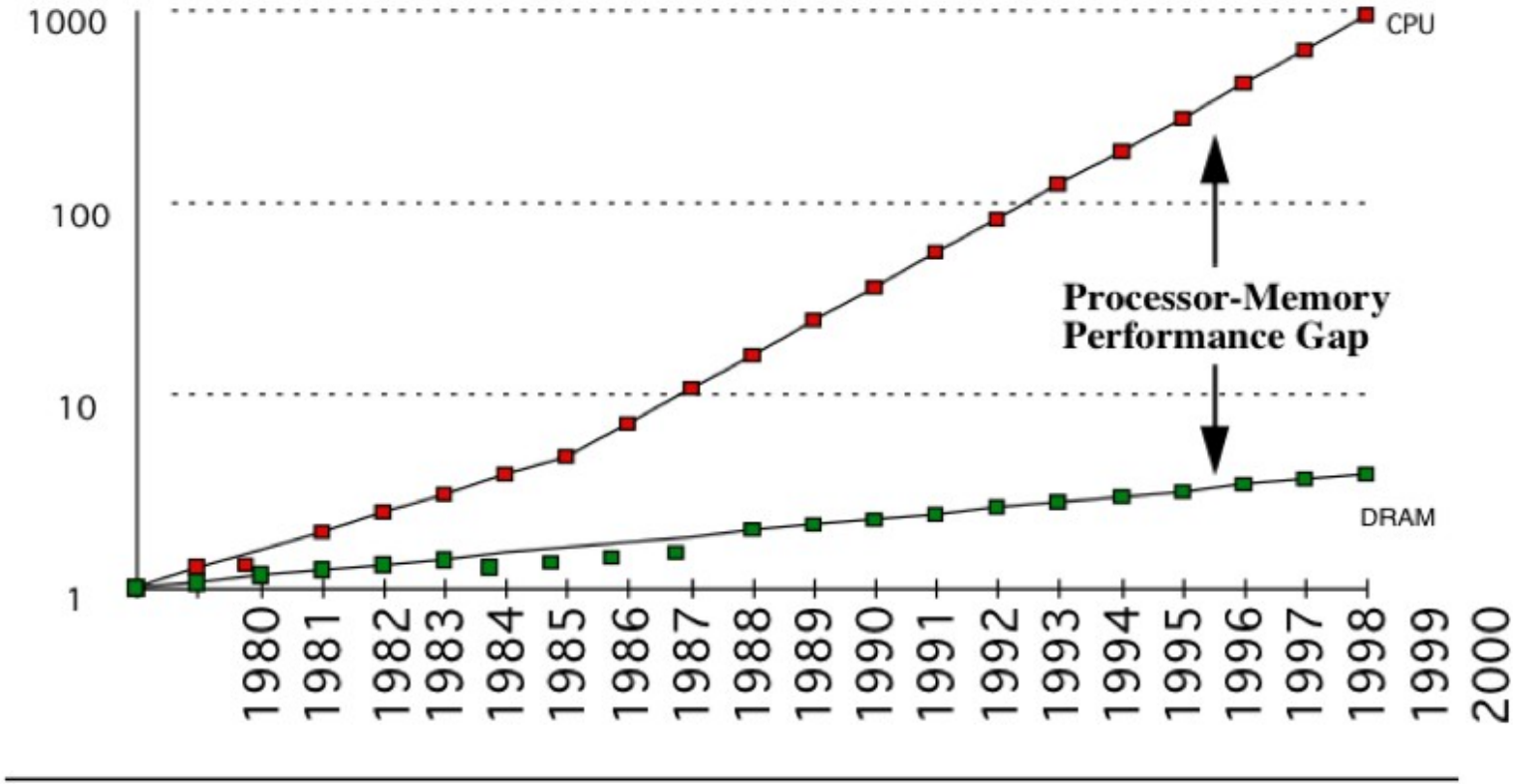
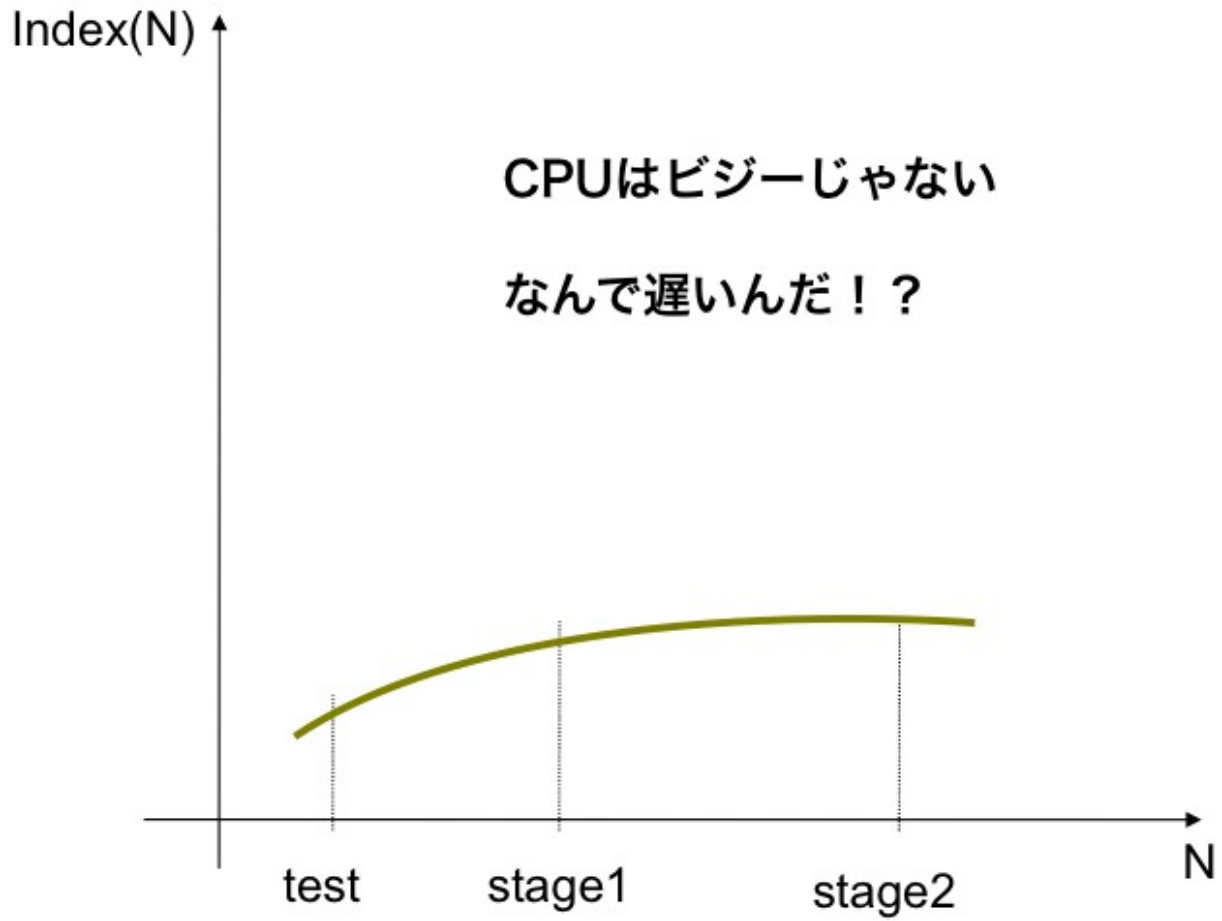


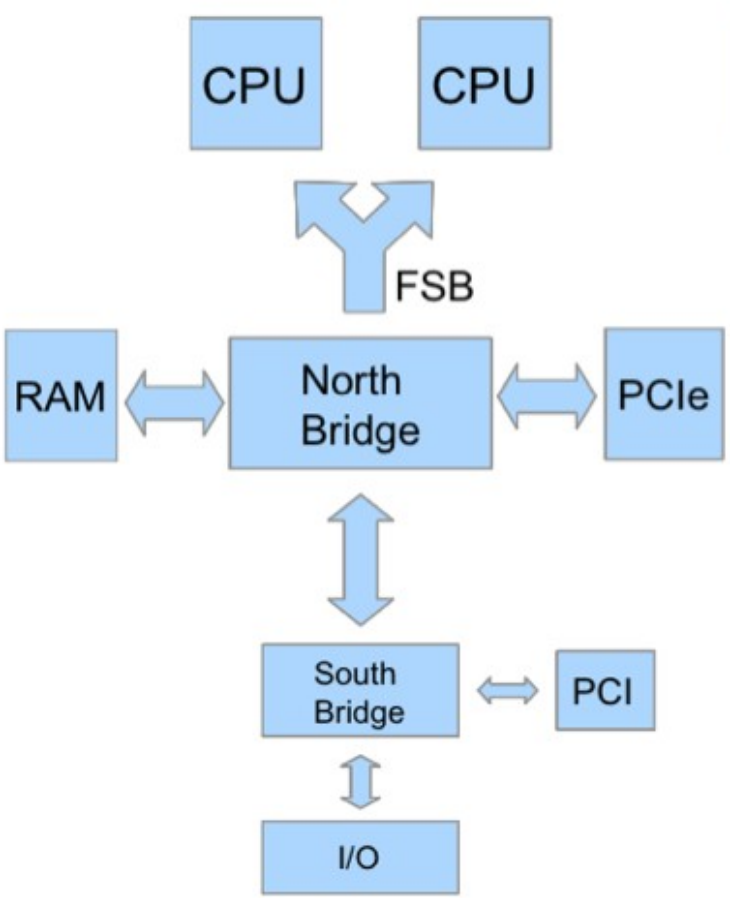
FIGURE 1. Processor-Memory Performance Gap.[Hen96].

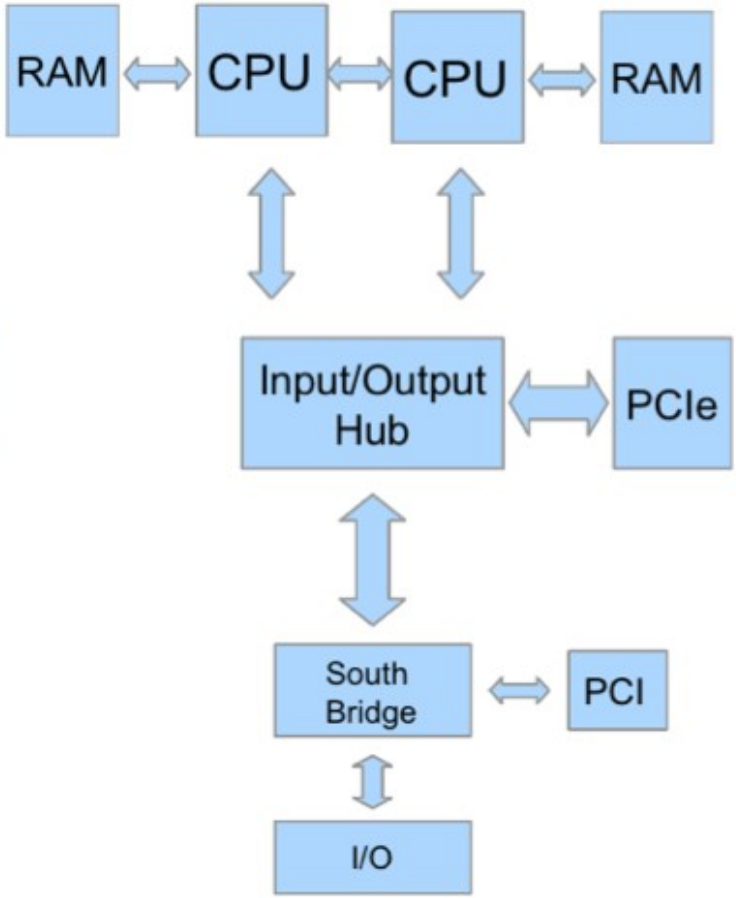
A Case for Intelligent RAM: IRAM, David Patterson他より

例えばグラフで２



対抗手段　CPUチップデザインの革新





対抗手段　ノーロックデザイン

RCU　リードコピーアップデート

ソフトウェアトランザクショナルメモリ

MVCC（マルチバージョン同時実行制御）

対抗手段　ノーロックデザインの特徴

リーダーは待つ事無しにアクセス可能

書き込みは１度に１つのみ

書き込みの性能を犠牲にしてもリード性能を向上

（データベースの用語だと）
Mutex: 悲観的ロックかつSerializable
No Lock: 楽観的ロックかつRead Committed

対抗手段　ノーロックデザインのポイント

1　更新はコピーして実行
2　更新されたコピーをアクセス可能にする
3　アクセスされなくなった古いバージョンを削除
　（どうやって知るか、いつ削除するか）

RCU　リードコピーアップデート

ソフトウェアトランザクショナルメモリ

MVCC（マルチバージョン同時実行制御）

対抗手段　ノーロックデザインの実装１

```
class CommandAttack : public CommandNode{
private:
	ssp_bushi enemy_;
public:
	CommandAttack(GameController *controller, ssp_bushi& bushi, ssp_bushi& enemy)
	  : CommandNode(controller, bushi), enemy_(enemy){

	}
	void execute(){
	  if(controller_->lock(enemy_) && controller_->lock(bushi_)){
		bool killed = bushi_->attack(enemy_);
		if(killed){
			std::cout << "getting rid of dead body" << std::endl;
			controller_->getSenjo()->out(enemy_->x(), enemy_->y(), enemy_->getId()); // could compete

			ssp_bushi left = controller_->getBushiAt(enemy_->x(), enemy_->y());
			if(!controller_->isBushiNull(left))
				std::cout << "the body seems to be left" << std::endl;
		}
	  }else{
		std::cout << "Giving up locking..." << std::endl;
	  }
	  controller_->unlock(bushi_);
	  controller_->unlock(enemy_);
	}
};
```

対抗手段　ノーロックデザインの実装使い方２

```
bool attack(ssp_bushi& enemy){
  if(!enemy->isAlive()){
        return false;
  }
  if(enemy->getTairyoku() <= kogekiryoku_){
        // enemy gets killed
        enemy->setTairyoku(0);

        kubi_list_.push_back(enemy);
        enemy->getButai()->killed(enemy);

        if(enemy->reward() == REWARD_SODAISHO){
              // stop the war
              GLOBAL_STATE = STATE_TERMINATED;
              std::cout << enemy->getName() << " Uchitottari! The war has finished..." << std::endl;
        }else if(enemy->reward() == REWARD_BUSHO){
              std::cout << enemy->getName() << " Uchitottari!" << std::endl;
        }
        return true;
  }else{
        enemy->setTairyoku(enemy->getTairyoku() - kogekiryoku_);
  }
  return false;
}
```

対抗手段　ノーロックデザインの実装使い方３

ファクトリー

武将

```
ssp_bushi nagamasa = sspService->create<Busho>(100002);
nagamasa->setName("Kuroda Nagamasa");
nagamasa->setX(520);
nagamasa->setY(500);
nagamasa->setId(100002);
nagamasa->setButai(all_butai_[0]);
```

足軽

```
ssp_bushi bushi = sspService->create<Ashigaru>(bushi_id);
bushi->setName("Togun Ashigaru " + itos(bushi_id));
bushi->setX(x);
bushi->setY(y);
bushi->setId(bushi_id);
bushi->setButai(all_butai_[butai_ix]);
```

グローバルアクセス

```
ssp_bushi getBushiAt(unsigned int x, unsigned int y){
  return sspService_->get(senjo_->getBushiIdAt(x, y));
}
```

対抗手段　ノーロックデザインの実装　内部１

```
// lock
bool lock(SSP<AbstractPointee, Storage>& ssp){

  if(ssp.locked_ ){
          std::cout << "already locked" << std::endl;
          return false;
  }

  if(ssp.storage_->lock()){
          if(ssp.ptr_ix_ != ssp.storage_->getCurrentPtrIndex()){
                  // this is a tricky situation.
                  // this instance refers to obsolete pointee
                  // lock should be made on the latest, so fix the situation
                  ssp.storage_->decrement(ssp.ptr_ix_);
                  ssp.ptr_ix_ = ssp.storage_->increment(); // should be equal to getCurrentPtrIndex
          }
          ssp.locked_ = true;
  }
  return ssp.locked_;
}
```

対抗手段　ノーロックデザインの実装　内部２

```
// any counting operations are not done until unlock is called
bool lock(){
 if(__sync_bool_compare_and_swap(&modified_, (T*)0, pointee_array_[ptr_ix_])){
        modified_ = pointee_array_[ptr_ix_]->clone();
        return true;
 }

 return false;
}
```

new Tでは不十分

対抗手段　ノーロックデザインの実装　内部３

```
bool unlock(SSP<AbstractPointee, Storage>& ssp){

  if(!ssp.locked_ ){
        return false;
  }

  unsigned int new_ptr_ix = ssp.storage_->unlock();
  if(new_ptr_ix < ssp.storage_->getMaxPtrSize()){
        if(new_ptr_ix != ssp.ptr_ix_){
              ssp.storage_->decrement(ssp.ptr_ix_);
              ssp.ptr_ix_ = new_ptr_ix;
        }
        ssp.locked_ = false;
        return true;
  }

  ssp.locked_ = false;
  return false;
}
```

対抗手段　ノーロックデザインの実装　内部４

```
unsigned int unlock(){

  unsigned int new_ptr_ix = findAvailableIndex();
  if(new_ptr_ix == ATOMIC_STORAGE_PTR_SIZE){
          // no slot availalbe
          delete modified_;
          modified_ = 0;
          std::cout << "no slots available" << std::endl;
          //printArray();
          return new_ptr_ix;
  }

  unsigned int old_ptr_ix = ptr_ix_;

  // assign new
  pointee_array_[new_ptr_ix] = modified_;
  counter_array_[new_ptr_ix] = 1;
  total_ix_++;
  // ================ full memory barrier ===============
  __sync_synchronize ();
  //
  ptr_ix_ = new_ptr_ix; // some readers may start reading
  modified_ = 0; // for writers

  return ptr_ix_;
}
```

ポイント

new_f->count = 3;
new_f->value = 0;

this->f = new_f;

* fのポインターの付け替えはatomicだが、他のコアまたはCPUからみた時countとvalueの値が更新されている保証はない

対抗手段　ノーロックデザインの実装　内部５

```
unsigned int findAvailableIndex(){
  unsigned int new_ptr_ix = 0;
  for(int i=1; i<ATOMIC_STORAGE_PTR_SIZE; i++){
          new_ptr_ix = (ptr_ix_+i >= ATOMIC_STORAGE_PTR_SIZE)?
                        (ptr_ix_ + i -ATOMIC_STORAGE_PTR_SIZE)
                        :ptr_ix_ + i;
          if(counter_array_[new_ptr_ix] == 0){
                  // found available slot

                  // lazy deletion
                  if(pointee_array_[new_ptr_ix])
                          deletePointer(new_ptr_ix);

                  return new_ptr_ix;
          }
  }

  return (unsigned int) ATOMIC_STORAGE_PTR_SIZE;
}
```

カウンターが０になった瞬間に削除しようとすると、ひどい同時実行問題に遭遇することがある

対抗手段　ノーロックデザインの実装　実力１

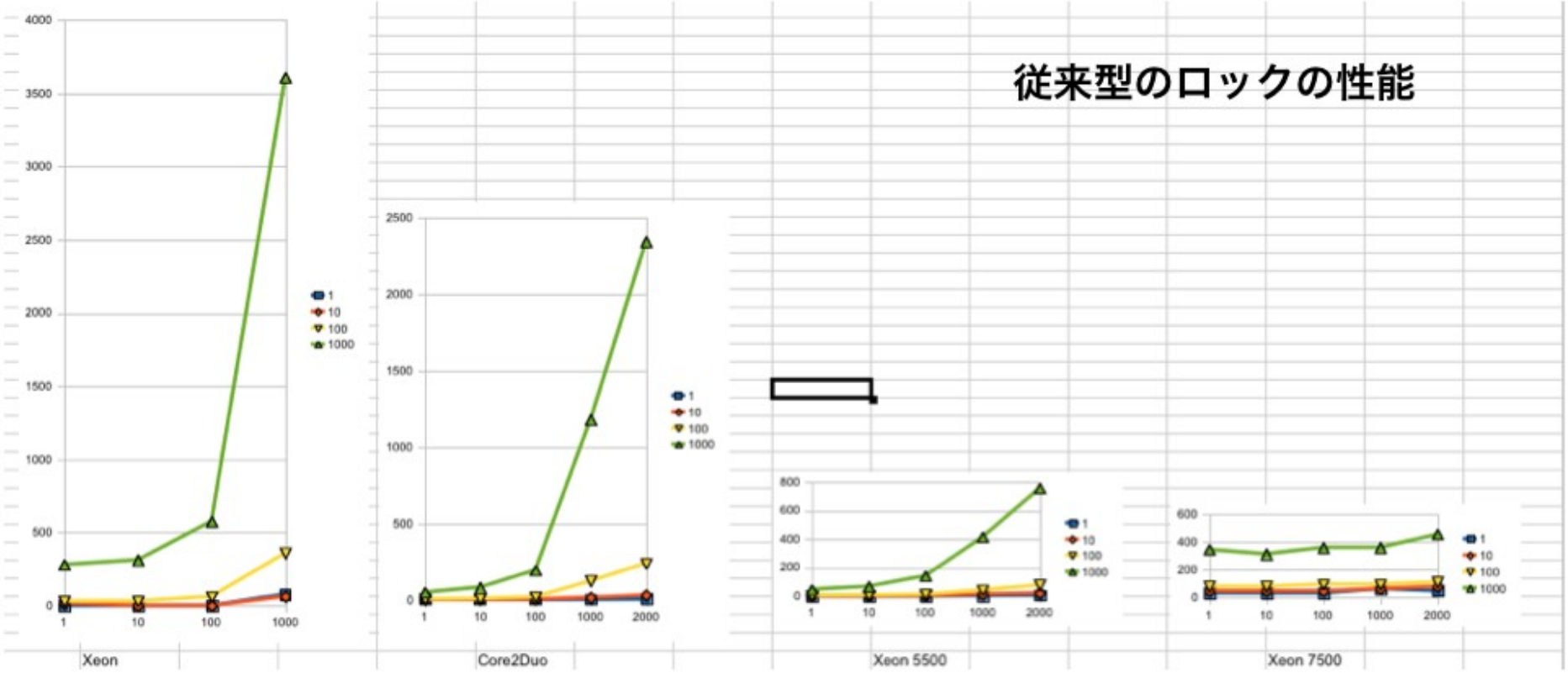
0. Development Windows
CPU: Intel® Xeon™ (1MB cache, 2.80GHz, 800MHz FSB) * 2
RAM: DDR2-800 Registered 512MB * 2, 256MB * 2
OS: Windows XP SP2

1. Development Linux
CPU: Intel® Core™2Duo P7370(3MB cache, 2.00GHz, 1066MHz FSB) * 1
RAM: DDR3-1066 2GB * 2
OS: CentOS 5.5(Linux Kernel 2.6.18)

2. Mid-Range
CPU: Intel® Xeon™ E5506(4MB cache, 2.13GHz, 4.8GT/s QPI) * 2
RAM: DDR3-1333 Registered 2GB * 4
OS: CentOS 5.5(Linux Kernel 2.6.18)

3. High-End (NEC Express5800/R140b-4)
CPU: Intel® Xeon™ X7560(24MB cache, 2.26GHz, 6.4GT/s QPI) * 4
RAM: DDR3-1066 Registered 8GB * 4
OS: Windows Server® 2008 R2

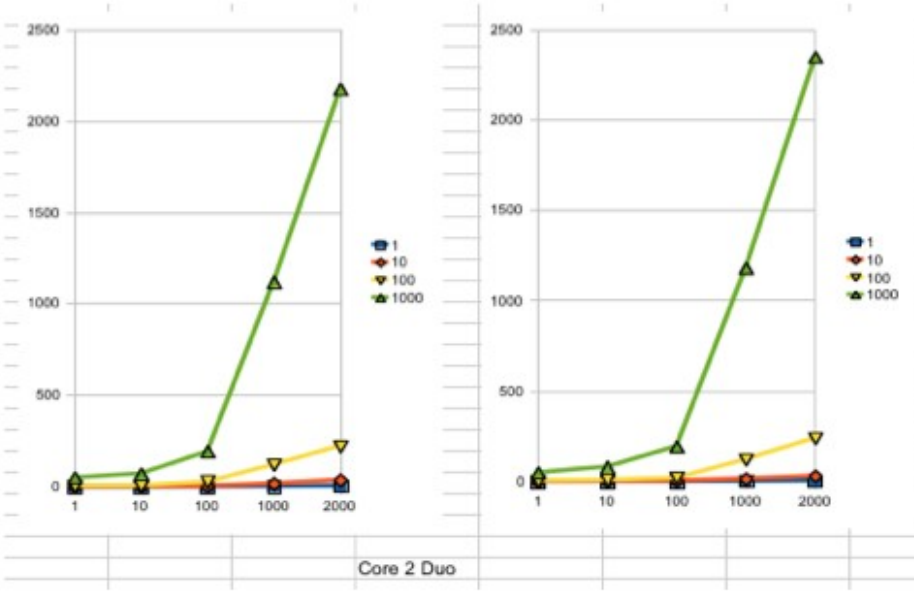
対抗手段　ノーロックデザインの実装　実力２



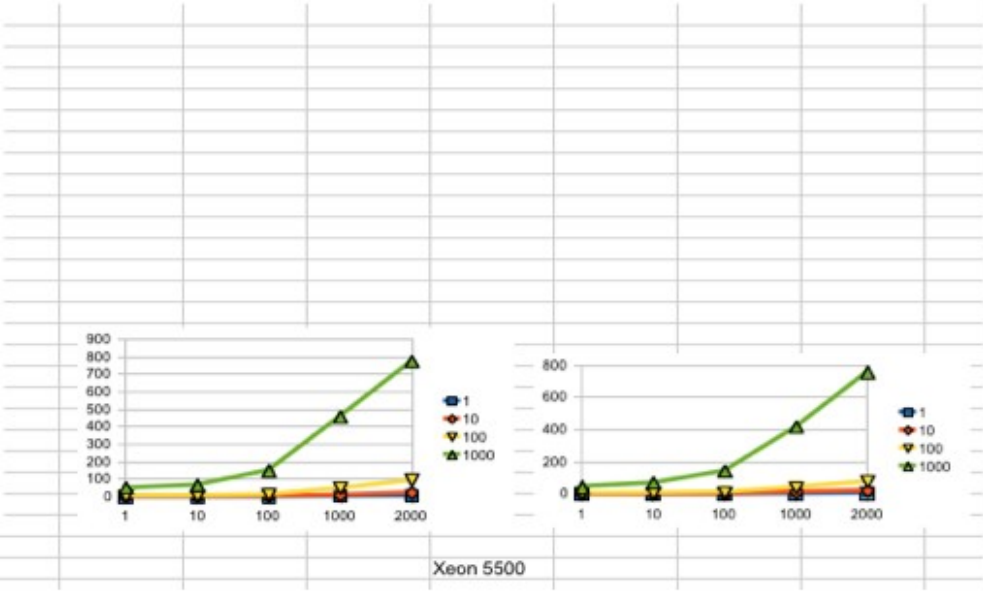
x: スレッド数　　y:所要時間　系列: ロックにかける長さ

対抗手段　ノーロックデザインの実装　実力３

左　ノーロック
右　従来のロック



Core 2 Duo

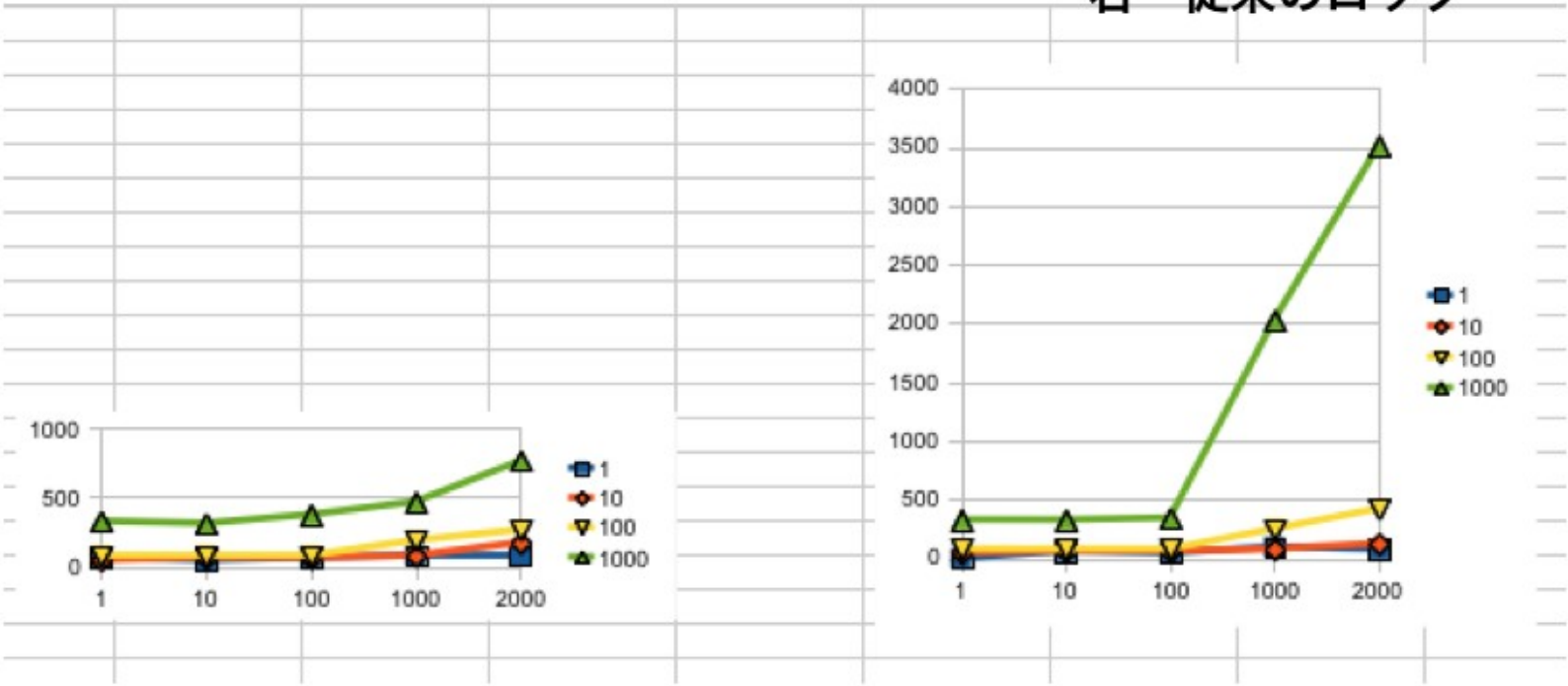


Xeon 5500

x: スレッド数　　y:所要時間　系列: ロックにかける長さ

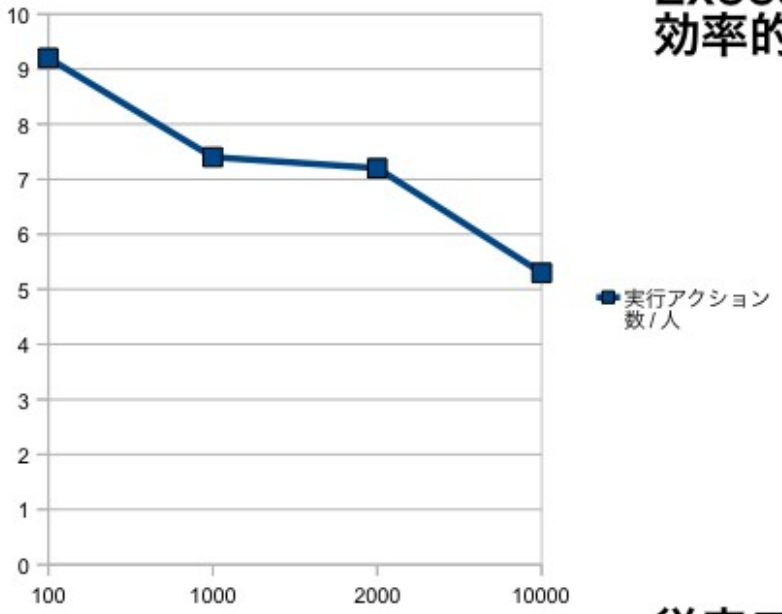
対抗手段　ノーロックデザインの実装　実力４

競合頻度を増やした場合
左　ノーロック
右　従来のロック



x: スレッド数　　y:所要時間　系列: ロックにかける長さ

関ヶ原シミュレーションでの検証



ExecutorとGeneratorの効率的なやり取りに課題

従来ロック版はデッドロックで採取不可

結論

デバッグが超大変
- gdb, valgrind, oprofileフル活用

共有メモリへのアクセスが競合する状況で効果を発揮
- 想定外のメリットとしては、Invalidateされるキャッシュラインが
  少なくなるので、相乗効果

ゲームプログラミングはすごい

連絡先

takenori@versant.com

Scalable Smart Pointer project page
http://code.google.com/p/ssptr/