SBCL + てけと一OpenMP

2012/MAY/12

たけおか@たけおかラボ(株)/AXE

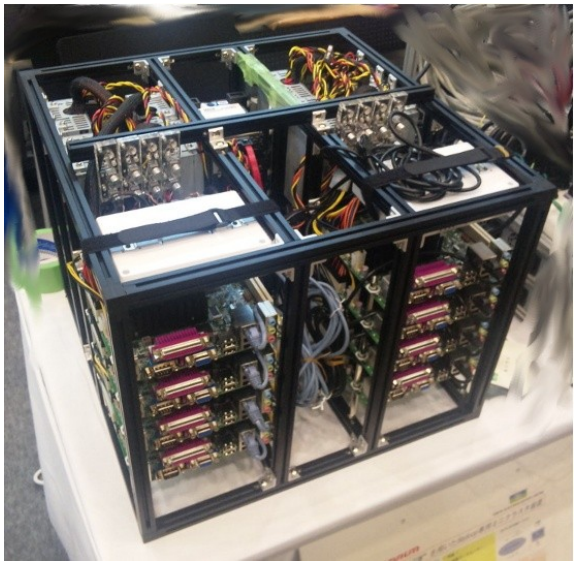
take@takelab.com

take@axe.bz

@takeoka

最近のあたくし

- BSDを愛する
  - BSDお仕事の会を発足
- スパコン,並列計算機
  - OSはLinux ;–<
- たけおかラボ(株)創業
  - 2012年2月14日
  - Lispでスパコン計算する



Hadoop専用クラスタ並列計算機

東京エレクトロンデバイス(TED)と共同開発中

Intel ATOM based parallel computer

32 core (16CPU)

Joint project with TED

Supporting hadoop

GASNet を「京」へポーティング。京の低レイヤ通信を使用

GASNet: Berkleyが開発した分散共有メモリ通信ミドルウェア

SBCLの
他言語インターフェース

バイナリはPIC, Shared

- SBCL起動後に、インクリメンタル・リンク
- 位置独立&共有ライブラリ形式であるべし

- gcc -shared -fPIC $(CFLAGS) ompvec.c -o ompvec.so

Cソースは、普通

```
void
vec_fmul(float *a, float *b, float *c,Int n)
{
  Int i,s;
  printf("vec_fmul = %d\n", n);
  clock_gettime(CLOCK_REALTIME, &tstart);
#pragma omp parallel
  {
#pragma omp for
   for(i = 0; i < n; i++)
     c[i] = a[i] * b[i];
  }
 clock_gettime(CLOCK_REALTIME, &tend);
 printf("time= %ld\n", tend.tv_nsec - tstart.tv_nsec );
 return;
}
```

OpenMP 指令 直後のブロックは並列に

OpenMP 指令
ループを分割し、
各スレッドで並列に実行

Cソースは、普通

```
float
vec_fsum(float *a,Int n)
{
  Int i;
  float s;
  printf("vec_fsum = %d\n", n);
  clock_gettime(CLOCK_REALTIME, &tstart);
   s=0.;
#pragma omp parallel
  {
#pragma omp for reduction(+:s)
   for(i = 0; i < n; i++) s += a[i];
  }
 clock_gettime(CLOCK_REALTIME, &tend);
 printf("time= %ld\n", tend.tv_nsec - tstart.tv_nsec );
 return s;
}
```

OpenMP 指令　直後のブロックは並列に

OpenMP 指令
ループを分割し、
各ループの合計を変数sへ

Lisp側 宣言,リンク

```
(define-alien-routine "vec_fsum" float
  (a (array  float nil))
  (n int))
(define-alien-routine "vec_fmul" void
  (a (array  float nil))
  (b (array  float nil))
  (c (array  float nil))
  (n int :in))
```

Cのグローバル変数も読み書きできる

```
(define-alien-variable "aho" int)
```

setq とか参照とか普通にできる。(型が合わないとエラー)

他言語バイナリのリンク 呼び出し前に行う

```
(load-shared-object "./ompvec.so")
```

てけとー
ベクトル計算
機能

Vector計算を付ける

- Cuda, OpenCLを使いたい
  - とりあえず、1次元配列(ベクトル)で
- しかし、CUDAのバイナリは、shared, PICにはならない

- 今日は、OpenMPで負けておいてやろか
  - つーか、負けてください

- 検算は、Common Lisp(SBCL)自身でやりました

Vector計算を付ける

- vec-fmul(a,b,c,n)
  - 配列a, 配列b の各要素を乗じて、配列cへ格納
  - 配列長はn

- vec-fadd(a,b,c,n)
  - 配列a, 配列b の各要素を加算して、配列cへ格納
  - 配列長はn

- vec-fsum(a,n)
  - 配列aの全要素を加算して、返す
  - 配列長はn

- ompthread n
  - openMPの総スレッド数をnに設定

Vector計算を付ける

- vec-mul(a,b,c,n)
  - 配列a, 配列b の各要素を乗じて、配列cへ格納
  - 配列長はn

- vec-add(a,b,c,n)
  - 配列a, 配列b の各要素を加算して、配列cへ格納
  - 配列長はn

- vec-sum(a,n)
  - 配列aの全要素を加算して、返す
  - 配列長はn

Vec-fmul

```
(defun fvm ()
  (let (x sum)
    (with-alien
     ((aa (array float 20000))
      (bb (array float 20000))
      (cc (array float 20000)))
     (setq *num* 20000)
   ;
     (dotimes (i *num*)
       (setf(deref aa i) (float (+ i 0)))
       (setf(deref bb i) (float (+ i 1))))
   ;
     (vec-fmul aa bb cc *num*)
     (setq sum(vec-fsum cc *num*))
   ;
     ;(dotimes (i *num*) (format t "~a " (deref cc i)))
     (format t "~%sum=~a " sum) )))
```

他言語呼び出しで使用する変数をバインドしつつ、手続きを

浮動小数点数の配列を3つ

配列を
初期化

vec-fmul

vec-fsum

Vecfmul 実行時間（nanosec）

Vec-fmul-avx-thread=4	Vec-fmul-avx-thread=2	Vec-fmul-avx-thread=1	Vec-fmul-noavx-thread=4	Vec-fmul-noavx-thread=2	Vec-fmul-noavx-thread=1
100089	60210	44855	2332289	65918	55359
105953	64112	45202	3874193	66801	56272
109369	72469	45397	4348628	67382	56322
137621	93668	45623	4718295	67523	56472
154409	95152	45623	4849137	67624	67428
201889	96071	45939	5112384	67970	69360
1076384	97064	51713	5275290	69675	69490

113258	65597	45439.8	4024508.4	67203	56106.25

- 実行したマシンが、2Core × 2Hyperthread
- Hyperthreadはブレが多い。しかも遅い
- 2Coreなので、2threadが確実に高速化
- AVX(Intelのベクトル機構)は、効いている: 高速になっている
- しかし、1スレッドが一番速い。ダメじゃん… （涙）

Vecfsum 実行時間(nanosec)

Vfsum-avx-thread=4	Vfsum-avx-thread=2	Vfsum-avx-thread=1	Vec-fsum-noavx-thread=4	Vec-fsum-noavx-thread=2	Vec-fsum-noavx-thread=1
20514	43240	78684	27369	43385	78765
25016	46545	78805	3274986	43406	78915
25072	46942	78851	4365521	43526	79201
25126	47609	79096	6830320	43551	80400
25232	47915	79954	7923113	43671	82096
23932	46084	78721	26534	43467	79320.25

- 2threadは1threadより速い。OpenMP化してよかった＼(^^)／
- AVXがあまり効いていない。1threadだと、AVXの効果はある

vec-fmulとvec-fsumの振る舞いの違い

- vec-fmul(a,b,c,n)
  - c[]=a[]*b[]
  - 2つの配列を読み、一つの配列へ書く
  - 3つの大きな範囲を舐める。キャッシュが効きにくい。
  - 全スレッド(2コア,4スレッド)が同時に、3つの配列のアクセス
  - 全スレッドが同時に2次キャッシュ・アクセス、メモリIOを行ってしまう(競合/待ち)

- vec-fsum(a,n)
  - Sum= Σ a[]
  - 配列はa[]のみ
    - 各スレッドはa[]の分担領域を読む
  - 合計は、各スレッドのレジスタ上に保持
  - 各スレッドは、最後に、共有変数に合計値を加算
  - キャッシュ&外部バスに配列1つのアクセスだけ。でも、全スレッド同時に起きている

- AVXは、メモリのロードを連続的に起こす
  - 1コアなら良い。2コア同時だと、待ち時間が増加しているのだろう(か?)

おしまい

www.takeoka.org/~take/