Orchestrating a brighter world NEC

衝突電離過程を含む相対論電磁粒子コードPICLSのSX-ACE向け最適化報告

2016/03/30　日本電気株式会社

もくじ

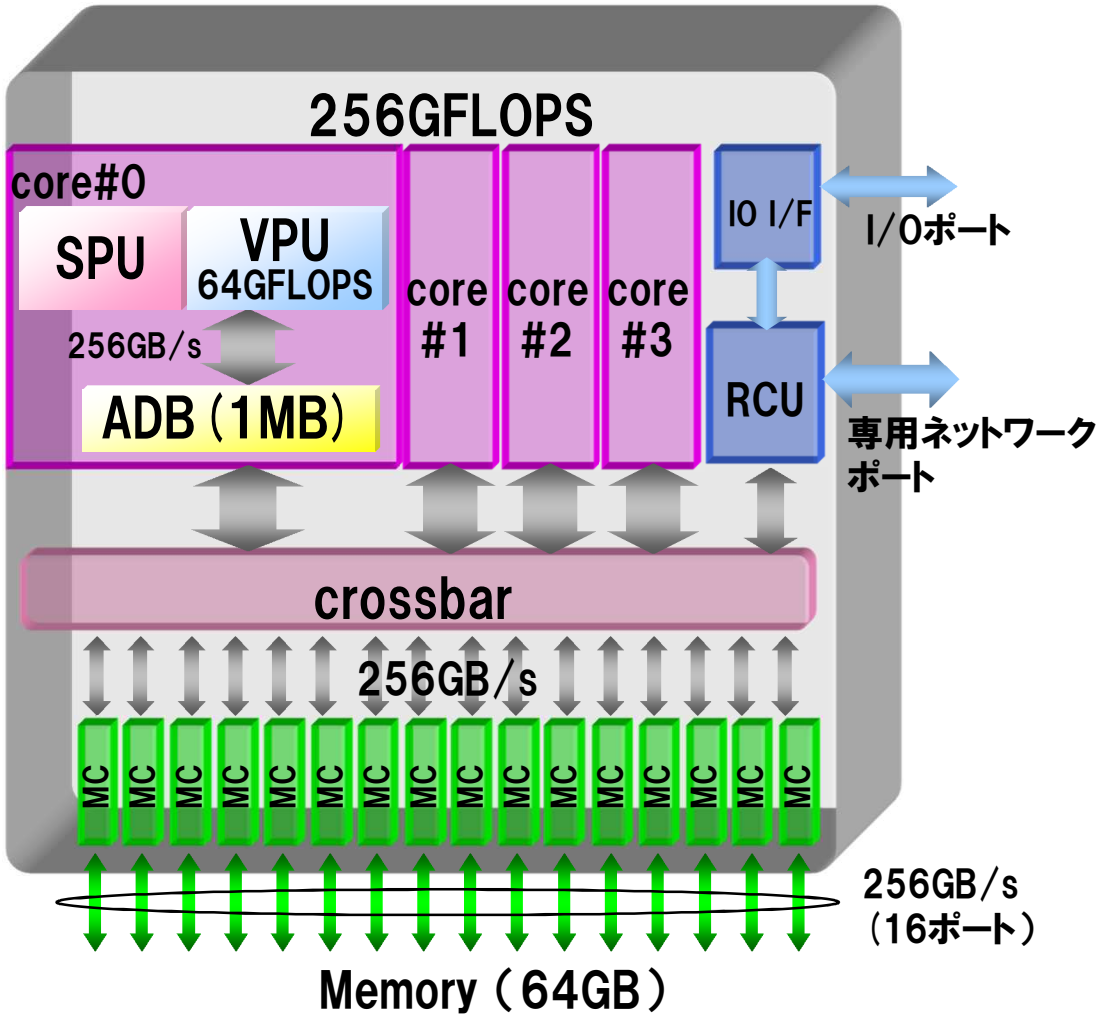
はじめに

はじめに

- 「衝突電離過程を含む相対論電磁粒子コードPICLS」というPICコードにて実施したSX-ACE向けベクトル化最適化について報告する.

SX-ACEのご紹介

- **演算性能　256GFLOPS（64GFLOPS/Core　×4Core）**
- **メモリバンド幅 256GB/s　（16GB/s /ポート　×16ポート）**



Core	
演算性能	64GFlops
ADBサイズ	1MB
ADB帯域	256GB/s
CPU	
Core数	4
演算性能	256GFlops
メモリ帯域	256GB/s
Byte/Flop	1

SPU: Scalar Processing Unit
VPU: Vector Processing Unit
ADB: Assignable Data Buffer
RCU: Remote Access Control Unit
MC: Memory Controller

ベクトル機向け最適化のポイント

ベクトル機の性能を引き出すためには

1. ループ長(ベクトル長)が十分に長い
2. ベクトル演算率が高い
3. メモリアクセス性能が高い

最適化効果(p_push)

p_push

do the "particle pushing"

(1) use the rotation matrix
(2) calculate the P^(n+1/2)

	行数	実行時間
修正前	約360行	411.03sec
修正後	約481行	24.30sec

×1/16.9

※約120行の修正および追加

```
FREQUENCY  EXCLUSIVE       AVER.TIME     MOPS    MFLOPS V.OP  AVER.    VECTOR I-CACHE O-CACHE   BANK CONFLICT    ADB HIT PROC.NAME
           TIME[sec](  %)     [msec]                    RATIO V.LEN      TIME    MISS    MISS CPU PORT  NETWORK   ELEM.%

      2001  411.026(  3.7)    205.410    858.4    308.0 62.98   6.8   252.343   0.011  23.708    0.000    0.491    99.77 p_push[org]
      2001   24.294(  0.5)     12.141  17141.8   5050.9 96.81 255.2    11.919   0.010   0.801    3.995    0.554    89.43 p_push[tune]
```

FLOPSとは…
1秒間に実行された浮動小数点演算数を示した値

最適化効果(p_push)

p_push

- 修正内容(詳細)

修正前

```
+------>      do 1000 is = 1,N_sp
 :
|+------>      do 1100 j = l_st(is),l_ed(is)
 :
||W*=== A       S0 = 0.0d0
  :
||          if(kpx.lt.0 .or. kpx.ge.nx .or. kpy.lt.0 .or. kpy.ge.ny_pe)
||        ¥     write(*,*) " out!! ",is,j,kpx,kpy
 :
||          if(is.eq.N_sp .and. rad_damping_opt) then
||             call rdamp(P(j,1),P(j,2),P(j,3),
||        ¥          p_mass(is),c,dlt_t,Exp,Eyp,Ezp,Bxp,Byp,Bzp,
||        ¥          rdamp_2nd_opt,coefr_1st,coefr_2nd)
||             call rdamp_spec(P(j,1),P(j,2),P(j,3),wgm(j),q(is),
||        ¥          p_mass(is),c,ow,dlt_t,Exp,Eyp,Ezp,Bxp,Byp,Bzp,
||        ¥          spec,Nw_rd,ang,Nang_rd,
||        ¥          rd_wmin,rd_wmax,dw_rd,dang_rd,iseed,npeon)
||          endif
|+-- 1100  continue
```

①

②

③

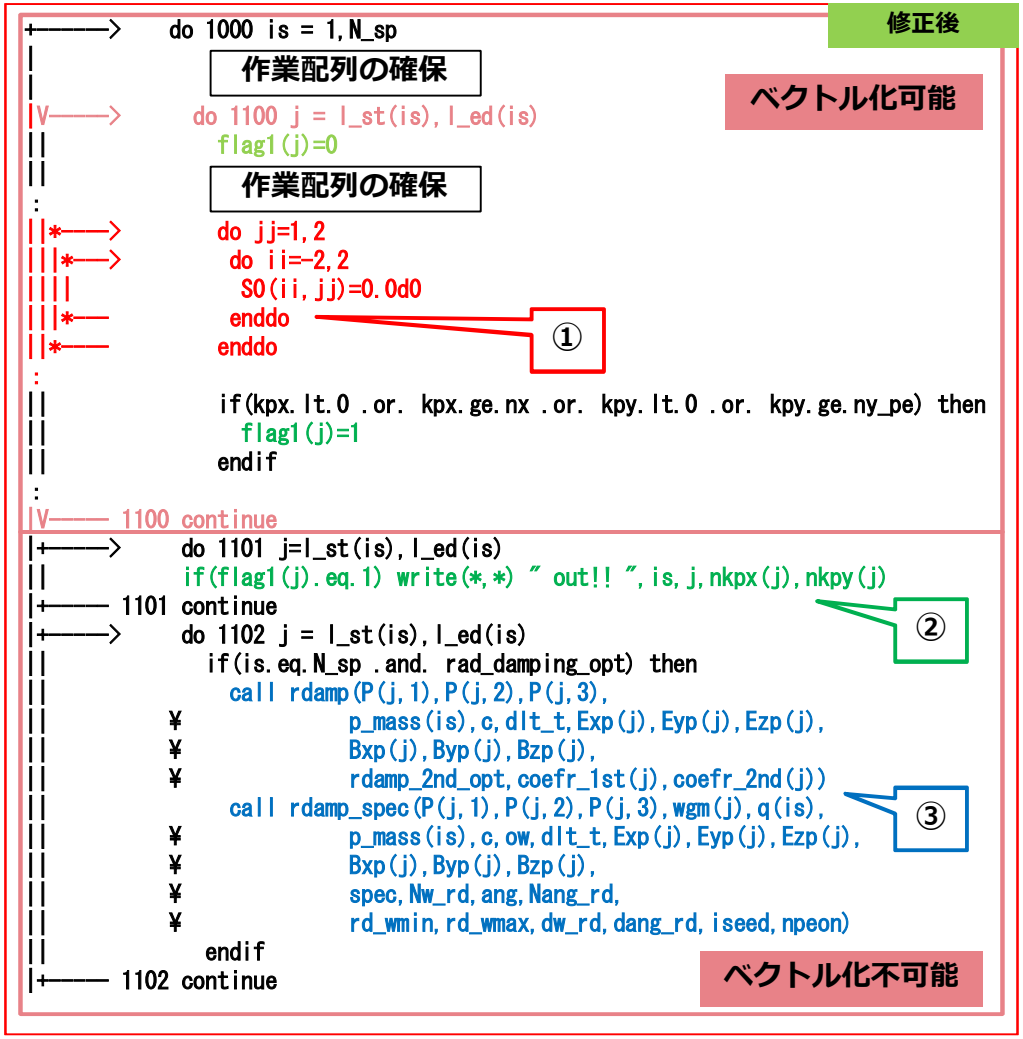
性能が低い原因…

①ベクトル長が短い(VL=10).
②write文を含むため，jのループでベクトル化できない.
③call文を含むため，jのループでベクトル化できない.

最適化効果(p_push)

p_push

- 修正内容(詳細)



高速化…

①最内ループを展開し，ループ長の大きいjのループでベクトル化を実施.
②write文をループ外へ移動.
③call文をループ外へ移動.

最適化効果(current_ab_rj_fi)

current_ab_rj_fi

charge conserved scheme & vector

- 修正内容
  - ベクトル化対象ループの変更によるベクトル長の拡大
  - メモリアクセス回数の削減
  - ベクトル化可能部分/不可能部分に分割しベクトル化可能部分のベクトル化を促進
  - 止まり木

※ベクトル化を阻害するため、デバッグ用write文は削除

	行数	実行時間
修正前	約400行	979.22sec
修正後	約650行	129.91sec

×1/7.5

※約150行の修正および追加

```
FREQUENCY  EXCLUSIVE        AVER.TIME     MOPS    MFLOPS V.OP  AVER.    VECTOR I-CACHE O-CACHE   BANK CONFLICT    ADB HIT PROC.NAME
           TIME[sec](  %  )    [msec]                     RATIO V.LEN     TIME    MISS    MISS CPU PORT  NETWORK   ELEM.%

      3002   979.214(  8.8)   326.187   3993.2    524.5 85.96  26.7  674.977   0.022  70.717    0.000  161.366    99.97 current_ab_rj_fi[org]
      3002   129.906(  2.6)    43.273  21791.0   6758.6 99.83 251.8  129.594   0.017   0.040    4.306   20.485    51.75 current_ab_rj_fi[tune]
```

最適化効果(current_ab_rj_fi)

current_ab_rj_fi

- 修正内容(詳細)

修正前

```
+-------> do 1500 is =  1,N_sp
|+------>    do 1600 jp = l_st(is), l_ed(is)
 :
||+------>   do i=-3,3
|||V----->      do j=-3,3
||||    A        WW(i,j,1) = DS(i,1)*(S0(j,2)+0.5d0*DS(j,2))
||||
||||    A        WW(i,j,2) = DS(j,2)*(S0(i,1)+0.5d0*DS(i,1))
||||
||||    A        WW(i,j,3) = S0(i,1)*S0(j,2) + DS(i,1)*S0(j,2)/2.d0
||||      ¥            + S0(i,1)*DS(j,2)/2.d0 + DS(i,1)*DS(j,2)/3.d0
|||V-----        enddo
||+------     enddo
||V------>   do j=-3,3
|||     A     SX(-3,j)=-WW(-3,j,1)*dlt_xg/dt*wgm0
||V------    enddo
||+------>   do i=-2,3
|||V----->     do j=-3,3
||||    A     SX(i,j)=SX(i-1,j)-WW(i,j,1)*dlt_xg/dt*wgm0
|||V-----     enddo
||+------    enddo
||V------>   do i=-3,3
|||     A     SY(i, -3)=-WW(i,-3,2)*dlt_yg/dt*wgm0
||V------    enddo
||X------>   do i=-3,3
|||+----->     do j=-2,3
||||    A     SY(i,j)=SY(i,j-1)-WW(i,j,2)*dlt_yg/dt*wgm0
|||+-----     enddo
||X------    enddo
||W------>   do j=-3,3
|||*----->      do i=-3,3
||||    A        SZ(i,j)=WW(i,j,3)*vz*wgm0
|||*-----       enddo
||W------    enddo
```

①

②

性能が低い原因…

①作業配列WWのメモリアクセス(ロード/ストア)が多発.
②ベクトル長が短い.

最適化効果(current_ab_rj_fi)

current_ab_rj_fi

- 修正内容(詳細)

修正後

```
+------>     do 1500 is =  1,N_sp
:
|V------>           do jp = l_st(is), l_ed(is)
||          !cdir expand=7
||*------>              do j=-3,3
|||    A                 SX(jp,-3,j)=-((S1(jp,-3,1)-S0(jp,-3,1))*(S0(jp,j,2)+0.5d0
|||         ¥            *(S1(jp,j,2)-S0(jp,j,2))))*dlt_xg/dt*wwgm0(jp)
||*------              enddo
||
||          !cdir unroll=6
||*------>              do i=-2,3
|||         !cdir expand=7
|||*------>                 do j=-3,3
||||    A                      SX(jp,i,j)=SX(jp,i-1,j)-((S1(jp,i,1)
||||         ¥                            -S0(jp,i,1))*(S0(jp,j,2)
||||         ¥                            +0.5d0*(S1(jp,j,2)-S0(jp,j,2))))
||||         ¥                            *dlt_xg/dt*wwgm0(jp)
|||*------                 enddo
||*------              enddo
||
:
||          !cdir unroll=7
||*------>              do j=-3,3
|||         !cdir expand=7
|||*------>                 do i=-3,3
||||    A                      SZ(jp,i,j)=(S0(jp,i,1)*S0(jp,j,2)
||||         ¥                            + (S1(jp,i,1)-S0(jp,i,1))
||||         ¥                            *S0(jp,j,2)/2.d0
||||         ¥                            + S0(jp,i,1)*(S1(jp,j,2)-S0(jp,j,2))/2.d0
||||         ¥                            + (S1(jp,i,1)-S0(jp,i,1))*(S1(jp,j,2)
||||         ¥                            -S0(jp,j,2))/3.d0)
||||         ¥                            *wvz(jp)*wwgm0(jp)
|||*------                 enddo
||*------               enddo
|V------              enddo
```

①

②

高速化…

①作業配列WWを用いず演算することでメモリアクセス数を削減.
②最内ループを展開し，ループ長の大きいjpのループでベクトル化を実施.

最適化効果(current_ab_rj_fi)

ベクトル化不可の依存関係

- 前の繰り返しで定義された値を参照する処理はベクトル化できない.

例

```
b(1)=1
b(2)=2
b(3)=1
b(4)=4
b(5)=5
b(6)=4
b(7)=3
 :
```

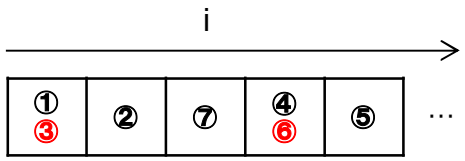
```
do i = 1,N
   list=b(i)
   a(list)=a(list)+wrk
enddo
```

間接参照(リストベクトル)

実行順序

```
a(1)=a(1)+wrk  ①
a(2)=a(2)+wrk  ②
a(1)=a(1)+wrk  ③
a(4)=a(4)+wrk  ④
a(5)=a(5)+wrk  ⑤
a(4)=a(4)+wrk  ⑥
a(3)=a(3)+wrk  ⑦
 :
```

③,⑥は前の繰り返しで定義された値を参照しているためベクトル化できない.



→「止まり木法」を用いることでベクトル化が可能に.

最適化効果(current_ab_rj_fi)

止まり木法

- 次元を１つ拡張した作業配列を作成し、重なりを無くすことでベクトル化が可能に。
- 作業配列を作成するためメモリ使用量が増加。処理量が増加。

例

```
b(1)=1
b(2)=2
b(3)=1
b(4)=4
b(5)=5
b(6)=4
b(7)=3
 :
```

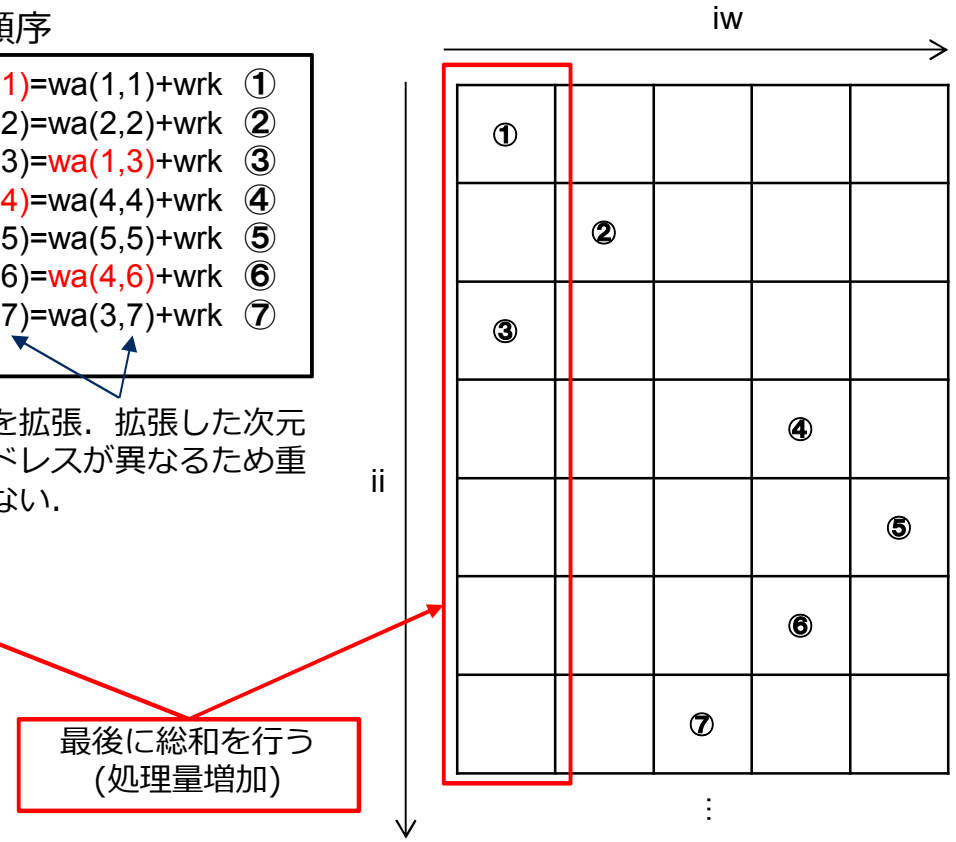
```
do iw= 1,N,256
 do ii =1,256
    i=iw+ii-1
    list=b(i)
    wa(list,ii)=wa(list,ii)+wrk
 enddo
enddo

do i=1,N
 do ii=1,256
   a(i)=a(i)+wa(i,ii)
 enddo
enddo
```

実行順序

```
wa(1,1)=wa(1,1)+wrk  ①
wa(2,2)=wa(2,2)+wrk  ②
wa(1,3)=wa(1,3)+wrk  ③
wa(4,4)=wa(4,4)+wrk  ④
wa(5,5)=wa(5,5)+wrk  ⑤
wa(4,6)=wa(4,6)+wrk  ⑥
wa(3,7)=wa(3,7)+wrk  ⑦
:
```

次元を拡張．拡張した次元のアドレスが異なるため重ならない．



最適化効果(current_ab_rj_fi)

current_ab_rj_fi

- 修正内容(詳細)

修正前

```
|+-----> do jp = l_st(is), l_ed(is)
||+----->    do j=-3,3
|||V----->    do i=-3,3                                    ①
||||           if(iflag(jp) .eq. 1)then
||||             coef = qc(jp)*qn(is)
||||             kx1  = nkx1(jp)
||||             ky1  = nky1(jp)
||||    A        rJx(i+kx1,j+ky1) =
||||        &         rJx(i+kx1,j+ky1) + SX(jp,i,j)*coef
||||    A        rJy(i+kx1,j+ky1) =
||||        &         rJy(i+kx1,j+ky1) + SY(jp,i,j)*coef
||||    A        rJz(i+kx1,j+ky1) =
||||        &         rJz(i+kx1,j+ky1) + SZ(jp,i,j)*coef
||||           endif
|||V----       enddo                                        ②
||+-----      enddo
|+------   enddo
```

性能が低い原因…

①ベクトル長が短い.
②jpのループでベクトル化したいがrJ[xyz]が間接参照であり、依存関係があるためにベクトル化できない.

最適化効果(current_ab_rj_fi)

current_ab_rj_fi

- 修正内容(詳細)

修正後

```
|W------>              do k = -3, Ny_pe+3
||*------>             do j = -3, Nx+3
|||          !cdir shortloop
||*------>             do i = 1, 256
||||                     wrkx(j,k,i) = 0.0d0
||||                     wrky(j,k,i) = 0.0d0
||||                     wrkz(j,k,i) = 0.0d0
||*-------             end do
|*--------              end do
W---------              end do
```

作業配列の初期化

止まり木法

```
|+------>           do jj = l_st(is), l_ed(is),256
||
||+------>            do j=-3,3
|||+------>           do i=-3,3
||||       !cdir shortloop
||||V-->               do ii = 1, min(256, l_ed(is)-jj+1)
|||||                    jp=jj+ii-1
|||||   A                if(iflag(jp) .eq. 1)then
|||||   A                  coef = qc(jp)*qn(is)
|||||                      kx1  = nkx1(jp)
|||||                      ky1  = nky1(jp)
|||||   A                  wrkx(i+kx1,j+ky1,ii) =
|||||         &              wrkx(i+kx1,j+ky1,ii) + SX(jp,i,j)*coef
|||||   A                  wrky(i+kx1,j+ky1,ii) =
|||||         &              wrky(i+kx1,j+ky1,ii) + SY(jp,i,j)*coef
|||||   A                  wrkz(i+kx1,j+ky1,ii) =
|||||         &              wrkz(i+kx1,j+ky1,ii) + SZ(jp,i,j)*coef
|||||                    endif
||||V-                  enddo
|||+--                 enddo
||+---                 enddo
|+----                enddo
```

⇒

修正後

```
|||          !cdir shortloop
|||+------>              do i = 1, 256
||||    A                 rJx(j,k) = rJx(j,k) + wrkx(j,k,i)
||||    A                 rJy(j,k) = rJy(j,k) + wrky(j,k,i)
||||    A                 rJz(j,k) = rJz(j,k) + wrkz(j,k,i)
|||+---                  end do
||X----                  end do
|+-----                  end do
```

総和

まとめ

- ループ長の短いループでベクトル化されていた．ループ長の長い外側のループでベクトル化することで平均ベクトル長が増加し性能が向上した.
- ベクトル化できていなかった箇所は，ベクトル化不可部分のループ外への移動や、止まり木法を用いることでベクトル演算率が上がり，性能も向上することができた.


Orchestrating a brighter world
NEC