Новосибирский государственный университет
Факультет информационных технологий
Кафедра параллельных вычислений

Эффективное программирование современных микропроцессоров и мультипроцессоров

Ручная векторизация программ (пример)

Преподаватели:
Киреев С.Е.
Калгин К.В.

Ручная векторизация программ

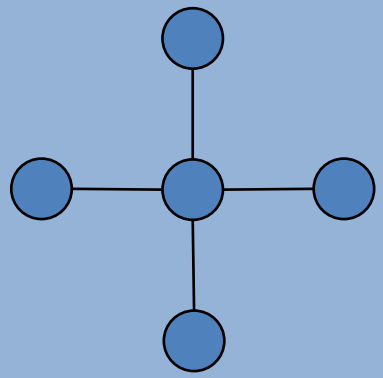
- Наиболее распространённое средство векторизации
  - Intel SIMD intrinsics
- Поддерживается почти всеми компиляторами
  - GNU, clang, Intel, Microsoft, …
- Типы данных
  - Целочисленные: __m64, __m128i, __m256i
  - Float: __m128, __m256
  - Double: __m128d, __m256d
- Операции (примеры)
  - __m64 _mm_add_pi32(__m64 a, __m64 b)
  - __m128 _mm_add_ss(__m128 a, __m128 b)
  - __m128 _mm_add_ps(__m128 a, __m128 b)
  - __m128d _mm_add_pd(__m128d a, __m128d b)
  - __m256d _mm256_add_pd(__m256d a, __m256d b)

Ручная векторизация программ

- Порядок действий
  - Выбрать цикл для векторизации
  - Выровнять в памяти массивы данных
  - Создать векторные аналоги адресов массивов
  - Создать векторные аналоги использующихся в цикле скалярных переменных (собрать вектора из скаляров)
  - Создать векторные аналоги вызываемых в цикле функций
  - Заменить скалярные операции на векторные
  - Поделить размеры массивов и циклов на размер вектора
  - Выделить при необходимости особые случаи в начале и конце цикла (возможно, частично векторизовать)

Ручная векторизация программ

- Пример:
  - Метод Якоби для решения 2D уравнения Пуассона
  - Используется схема «крест»



Ручная векторизация программ

- Самая «тяжёлая» часть программы

```
double f(int i,int j) { return 4.0; }

double step(double *u1,double *u2,int it)
{ double rhx2 = 1.0/(hx*hx);
  double rhy2 = 1.0/(hy*hy);
  double kf = 0.5/(rhx2+rhy2);
  double delta=0;
  int i,j;
  for (i=1;i<NY-1;i++)
  { for (j=1;j<NX-1;j++)
    { int ij = i*NX+j;
      u2[ij] = kf*(rhx2*(u1[ij-1]+u1[ij+1])+rhy2*(u1[ij-NX]+u1[ij+NX])-f(i,j));
      double d = fabs(u1[ij]-u2[ij]);
      if (d>delta) delta = d;
    }
  }
  return delta;
}
```

Ручная векторизация программ

- Подготовка к векторизации:

  Создадим векторные аналоги функций, адресов и скаляров

```
double f(int i,int j) { return 4.0; }
__m128d vf(int i,int j) { return _mm_set1_pd(4.0); }

double step(double *u1,double *u2,int it)
{  __m128d *vu1 = (__m128d*)u1;
   __m128d *vu2 = (__m128d*)u2;
   double rhx2 = 1.0/(hx*hx);     __m128d vrhx2 = _mm_set1_pd(rhx2);
   double rhy2 = 1.0/(hy*hy);     __m128d vrhy2 = _mm_set1_pd(rhy2);
   double kf = 0.5/(rhx2+rhy2); __m128d vkf   = _mm_set1_pd(kf  );
   double delta=0;                __m128d vdelta = _mm_setzero();
   int i,j;
   for (i=1;i<NY-1;i++)
   { for (j=1;j<NX-1;j++)
     { int ij = i*NX+j;
       u2[ij] = kf*(rhx2*(u1[ij-1]+u1[ij+1])+rhy2*(u1[ij-NX]+u1[ij+NX])-f(i,j));
       double d = fabs(u1[ij]-u2[ij]);
       if (d>delta) delta = d;
     }
   }
   return delta;
}
```

Ручная векторизация программ

- Подготовка к векторизации: выравнивание
  - Было

```
double *u1 = (double *)malloc(2*NX*NY*sizeof(double));
double *u2 = u1 + NX*NY;

time_start(&tv);
count(u1,u2);
t=time_stop(&tv);
```

  - Стало:

```
if (NX%2!=0) { printf("Alignment error: NX = %d\n",NX); exit(1); }

double *u1;
if (posix_memalign((void**)&u1,16,2*NX*NY*sizeof(double))!=0) exit(1);
double *u2 = u1 + NX*NY;

time_start(&tv);
count(u1,u2);
t=time_stop(&tv);
```

Ручная векторизация программ

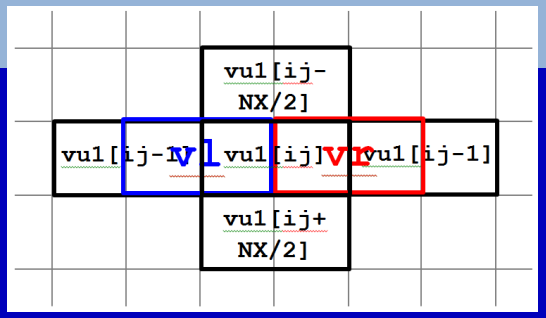
- Рассмотрим векторизуемый цикл

```
double f(int i,int j) { return 4.0; }
__m128d vf(int i,int j) { return _mm_set1_pd(4.0); }

double step(double *u1,double *u2,int it)
{  __m128d *vu1 = (__m128d*)u1;
   __m128d *vu2 = (__m128d*)u2;
   double rhx2 = 1.0/(hx*hx);     __m128d vrhx2 = _mm_set1_pd(rhx2);
   double rhy2 = 1.0/(hy*hy);     __m128d vrhy2 = _mm_set1_pd(rhy2);
   double kf = 0.5/(rhx2+rhy2); __m128d vkf   = _mm_set1_pd(kf  );
   double delta=0;                __m128d vdelta = _mm_setzero();
   int i,j;
   for (i=1;i<NY-1;i++)
   { for (j=1;j<NX-1;j++)
     { int ij = i*NX+j;
       u2[ij] = kf*(rhx2*(u1[ij-1]+u1[ij+1])+rhy2*(u1[ij-NX]+u1[ij+NX])-f(i,j));
       double d = fabs(u1[ij]-u2[ij]);
       if (d>delta) delta = d;
     }
   }
   return delta;
}
```

Ручная векторизация программ

- Перепишем цикл в векторной форме
  - Поделим число элементов на размер вектора
  - Заменим переменные на векторные аналоги
  - Обозначим недостающие данные и вычисления

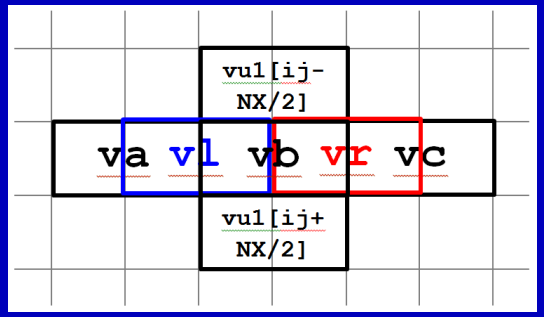



```
for (i=1;i<NY-1;i++)
{ ...
  for (j=1;j<NX/2-1;j++)
  { int ij = i*NX/2+j;
    __m128d vl = ???
    __m128d vr = ???
    vu2[ij] = vkf*(vrhx2*(vl+vr)+vrhy2*(vu1[ij-NX/2]+vu1[ij+NX/2])-vf(i,j));
    __m128d vd = fabs(vu1[ij]-vu2[ij]);
    if (vd>vdelta) vdelta = vd;
  }
  ...
}
```

Ручная векторизация программ

- Сделаем оптимизацию по памяти
  - Уберём лишние обращения к массиву

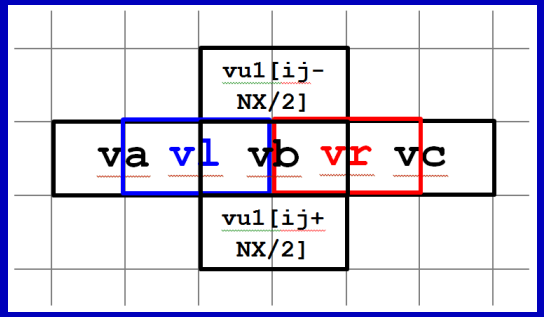
```
for (i=1;i<NY-1;i++)
{ ...
  __m128d va = vu1[i*NX/2+0];
  __m128d vb = vu1[i*NX/2+1];
  for (j=1;j<NX/2-1;j++)
  { int ij = i*NX/2+j;
    __m128d vc = vu1[ij+1];
    __m128d vl = ??? (va,vb);
    __m128d vr = ??? (vb,vc);
    vu2[ij] = vkf*(vrhx2*(vl+vr)+vrhy2*(vu1[ij-NX/2]+vu1[ij+NX/2])-vf(i,j));
    __m128d vd = fabs(vb-vu2[ij]);
    if (vd>vdelta) vdelta = vd;
    va = vb; vb = vc;
  }
  ...
}
```



Ручная векторизация программ

- Сделаем оптимизацию по памяти
  - Ещё уберём лишние обращения к массиву

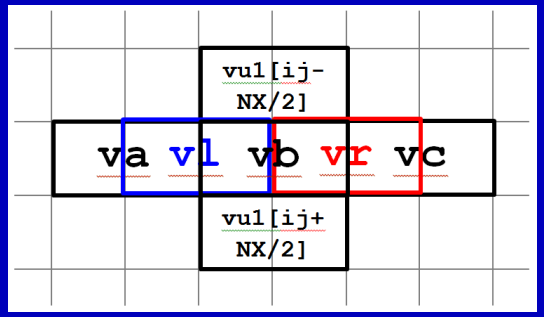
```
for (i=1;i<NY-1;i++)
{ ...
  __m128d va = vu1[i*NX/2+0];
  __m128d vb = vu1[i*NX/2+1];
  for (j=1;j<NX/2-1;j++)
  { int ij = i*NX/2+j;
    __m128d vc = vu1[ij+1];
    __m128d vl = ??? (va,vb);
    __m128d vr = ??? (vb,vc);
    __m128d v2 = vkf*(vrhx2*(vl+vr)+vrhy2*(vu1[ij-NX/2]+vu1[ij+NX/2])-vf(i,j));
    vu2[ij] = v2;
    __m128d vd = fabs(vb-v2);
    if (vd>vdelta) vdelta = vd;
    va = vb; vb = vc;
  }
  ...
}
```



Ручная векторизация программ

- Вычислим элементы vl и vr

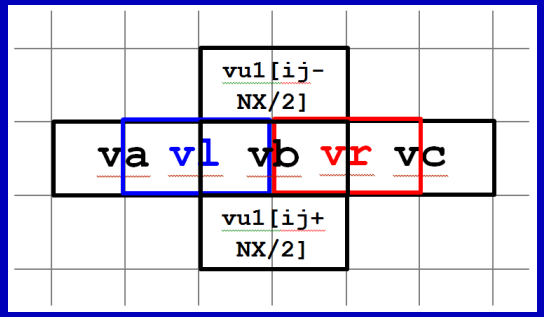
```
for (i=1;i<NY-1;i++)
{ ...
  __m128d va = vu1[i*NX/2+0];
  __m128d vb = vu1[i*NX/2+1];
  for (j=1;j<NX/2-1;j++)
  { int ij = i*NX/2+j;
    __m128d vc = vu1[ij+1];
    __m128d vl = _mm_shuffle_pd(va,vb,1);
    __m128d vr = _mm_shuffle_pd(vb,vc,1);
    __m128d v2 = vkf*(vrhx2*(vl+vr)+vrhy2*(vu1[ij-NX/2]+vu1[ij+NX/2])-vf(i,j));
    vu2[ij] = v2;
    __m128d vd = fabs(vb-v2);
    if (vd>vdelta) vdelta = vd;
    va = vb; vb = vc;
  }
  ...
}
```



Ручная векторизация программ

- Уберём лишние вычисления из цикла

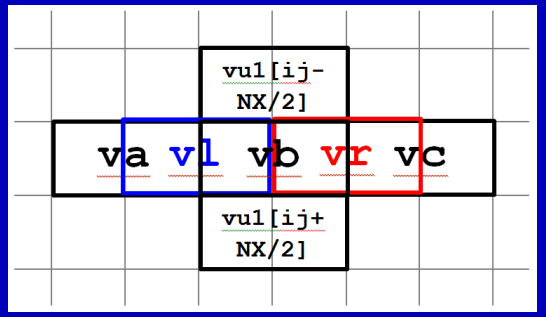
```
for (i=1;i<NY-1;i++)
{ ...
  __m128d va = vu1[i*NX/2+0];
  __m128d vb = vu1[i*NX/2+1];
  __m128d vl = _mm_shuffle_pd(va,vb,1);
  for (j=1;j<NX/2-1;j++)
  { int ij = i*NX/2+j;
    __m128d vc = vu1[ij+1];
    __m128d vr = _mm_shuffle_pd(vb,vc,1);
    __m128d v2 = vkf*(vrhx2*(vl+vr)+vrhy2*(vu1[ij-NX/2]+vu1[ij+NX/2])-vf(i,j));
    vu2[ij] = v2;
    __m128d vd = fabs(vb-v2);
    if (vd>vdelta) vdelta = vd;
    va = vb; vb = vc; vl = vr;
  }
  ...
}
```



Ручная векторизация программ

- Посчитаем vdelta = max(vdelta,|vb-v2|)

```
for (i=1;i<NY-1;i++)
{ ...
  __m128d va = vu1[i*NX/2+0];
  __m128d vb = vu1[i*NX/2+1];
  __m128d vl = _mm_shuffle_pd(va,vb,1);
  for (j=1;j<NX/2-1;j++)
  { int ij = i*NX/2+j;
    __m128d vc = vu1[ij+1];
    __m128d vr = _mm_shuffle_pd(vb,vc,1);
    __m128d v2 = vkf*(vrhx2*(vl+vr)+vrhy2*(vu1[ij-NX/2]+vu1[ij+NX/2])-vf(i,j));
    vu2[ij] = v2;
    __m128d vd = _mm_max_pd(vb-v2,v2-vb);
    __m128d vdelta = _mm_max_pd(vdelta,vd);
    va = vb; vb = vc; vl = vr;
  }
  ...
}
```



Ручная векторизация программ

- Добавим обработку первого и последнего элементов

```
for (i=1;i<NY-1;i++)
{ int ij = i*NX+1;
  u2[ij] = kf*(rhx2*(u1[ij-1]+u1[ij+1]) + rhy2*(u1[ij-NX]+u1[ij+NX]) - f(i,j));
  double d = fabs(u1[ij]-u2[ij]);
  if (d>delta) delta = d;
  __m128d va = vu1[i*NX/2+0];
  __m128d vb = vu1[i*NX/2+1];
  __m128d vl = _mm_shuffle_pd(va,vb,1);
  for (j=1;j<NX/2-1;j++)
  { ij = i*NX/2+j;
    __m128d vc = vu1[ij+1];
    __m128d vr = _mm_shuffle_pd(vb,vc,1);
    __m128d v2 = vkf*(vrhx2*(vl+vr)+vrhy2*(vu1[ij-NX/2]+vu1[ij+NX/2])-vf(i,j));
    vu2[ij] = v2;
    __m128d vd = _mm_max_pd(vb-v2,v2-vb);
    __m128d vdelta = _mm_max_pd(vdelta,vd);
    va = vb; vb = vc; vl = vr;
  }
  ij = i*NX+NX-2;
  u2[ij] = kf*(rhx2*(u1[ij-1]+u1[ij+1]) + rhy2*(u1[ij-NX]+u1[ij+NX]) - f(i,j));
  d = fabs(u1[ij]-u2[ij]);
  if (d>delta) delta = d;
}
```

Ручная векторизация программ

- Посчитаем delta = max(vdelta, delta)

```
      __m128d v2 = vkf*(vrhx2*(vl+vr)+vrhy2*(vu1[ij-NX/2]+vu1[ij+NX/2])-vf(i,j));

      vu2[ij] = v2;
      __m128d vd = _mm_max_pd(vb-v2,v2-vb);
      __m128d vdelta = _mm_max_pd(vdelta,vd);
      va = vb; vb = vc; vl = vr;
    }
    ij = i*NX+NX-2;
    u2[ij] = kf*(rhx2*(u1[ij-1]+u1[ij+1]) + rhy2*(u1[ij-NX]+u1[ij+NX]) - f(i,j));
    d = fabs(u1[ij]-u2[ij]);
    if (d>delta) delta = d;
  }
  __m128d vd = _mm_shuffle_pd(vdelta,vdelta,1);
  vdelta = _mm_max_sd(vdelta,vd);
  vdelta = _mm_max_sd(vdelta,_mm_set_sd(delta));
  _mm_store_sd(&delta,vdelta);
  return delta;
}
```

Ручная векторизация программ

- Используем команды FMA (при наличии)
  - Было:

```
__m128d v2 = vkf*(vrhx2*(vl+vr)+vrhy2*(vu1[ij-NX/2]+vu1[ij+NX/2])-vf(i,j));
```

  - Стало:

```
__m128d v2 = vkf*_mm_fmadd_pd(vrhx2,vl+vr,
                              _mm_fmsub_pd(vrhy2,vu1[ij-NX/2]+vu1[ij+NX/2],vf(i,j))
                             );
```

Ручная векторизация программ

- Сравнение времени работы

Версия программы	Оптимизация компилятором	Ручная оптимизация
Без изменений	6.19 с	
SSE	6.10 с	4.03 с
SSE + FMA		3.63 с
AVX	6.00 с	2.88 с
AVX + FMA	6.06 с	2.62 с