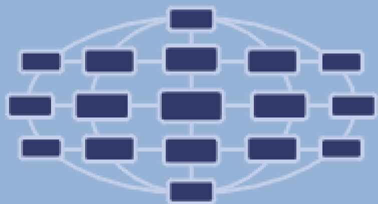


Новосибирский государственный университет
Факультет информационных технологий
Кафедра параллельных вычислений

Эффективное программирование современных микропроцессоров и мультипроцессоров

Векторизация вычислений

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

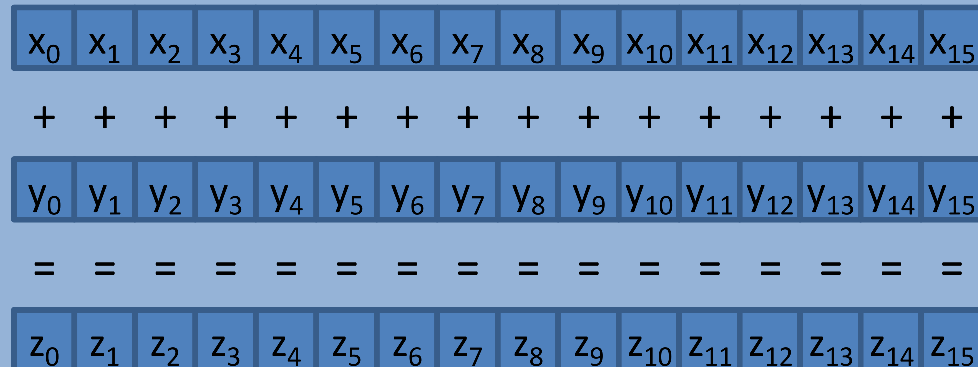
План лекции

- Сведения из архитектуры
- Цель векторизации
- Проблемы векторизации
- Средства векторизации
- Автоматическая векторизация в Intel C/C++ Compiler

СВЕДЕНИЯ ИЗ АРХИТЕКТУРЫ

Сведения из архитектуры

- **Векторные вычисления** –
  это вид параллельных вычислений
  с параллелизмом на уровне данных
  (SIMD – Single Instruction Multiple Data)



Сведения из архитектуры

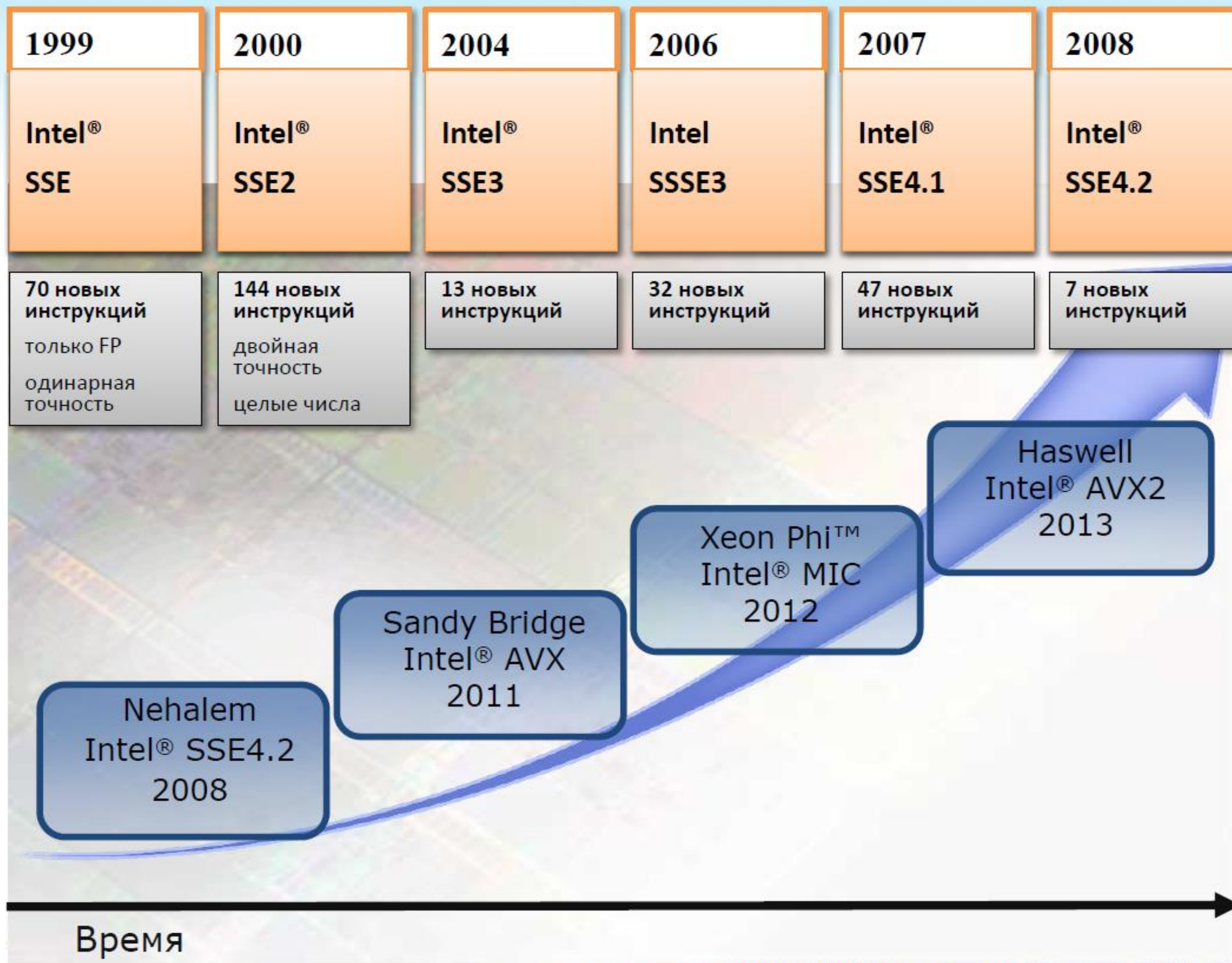
- В современных скалярных микропроцессорах общего назначения векторные вычисления поддержаны с помощью **векторных расширений** архитектуры
    - Примеры векторных расширений: MMX, SSE, AVX, …
- Векторные расширения включают:
  - Векторные регистры – хранят множества скалярных значений
    - Примеры: mm0-mm7, xmm0-xmm15, ymm0-ymm15
  - Векторные команды – для работы с векторными регистрами
    - paddb mm1,mm3; mulpd xmm0, xmm1
- Особенности векторных команд работы с памятью:
  - Существуют команды выровненного и невыровненного обращения к памяти
  - Команды выровненного обращения требуют выровненного адреса и работают максимально быстро
  - Команды невыровненного обращения могут обращаться по любому адресу, но могут работать медленнее

Сведения из архитектуры

Векторные расширения архитектуры x86

- Расширение MMX, 3DNow!, MMX Extended – Pentium MMX, K6-2
  - Регистры: 8 × 8 байт
  - Типы данных: целочисленные, float
  - Базовые операции, арифметика с насыщением, двухоперандные операции: x += y
- Расширение SSE – Pentium III, Athlon XP
  - Регистры: 8 (16) × 16 байт
  - Типы данных: float
  - Базовые операции
- Расширения SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, SSE4a – Pentium 4, Opteron, …
  - Регистры: 8 (16) × 16 байт
  - Типы данных: целочисленные, float, double
  - +Горизонтальные операции, специальные операции
- Специальные расширения: CLMUL, AES, XOP, F16C, FMA4, FMA3, …
- Расширение AVX – Ivy Bridge, Bulldozer
  - Регистры: 16 × 32 байта
  - Типы данных: float, double
  - Трёхоперандные неразрушающие операции: z = x + y
  - +Отсутствуют требования к выравниванию (но это влияет на производительность)
- Расширение AVX2 – Haswell
  - Регистры: 16 × 32 байта
  - Типы данных: целочисленные, float, double
- Расширение AVX-512 – Xeon Phi
  - Регистры: 16 × 64 байта
  - Типы данных: целочисленные, float, double

Поколения SSE



intel Software

© 2013, Intel Corporation. All rights reserved. Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others.

OPTIMIZATION NOTICE

ЦЕЛЬ ВЕКТОРИЗАЦИИ

Цель векторизации

- **Скалярная программа** – программа, оперирующая отдельными числами
- **Векторная программа** – программа, оперирующая векторами
- **Векторизация** (вид распараллеливания) – преобразование скалярной программы в векторную

Цель векторизации

- Цели
  1. **Ускорить работу программы**
  2. Уменьшить объем кода

- Предпосылки
  – Одна векторная команда распознаётся, декодируется и выполняется быстрее нескольких скалярных, выполняющих то же действие
  – Одна векторная команда занимает меньше места в программе и в различных очередях/таблицах/буферах в процессоре

ПРОБЛЕМЫ ВЕКТОРИЗАЦИИ

Проблемы векторизации

- Поиск в программе однотипных операций над различными данными
  (приведение к однотипным операциям)
    - Проще для операций с векторами и массивами
- Доказательство независимости операций
- Выровненный доступ к данным
- Оценка затрат на сборку-разборку векторов
    - Выигрыш должен быть больше затрат
- Переносимость
    - Какое векторное расширение использовать?
    - Многоверсионный код

СРЕДСТВА ВЕКТОРИЗАЦИИ

Средства векторизации

- Вставки на ассемблере (микрокодирование)
- Векторные операции и типы данных в языке
    - Встроенные в компилятор операции (intrinsics) и типы данных
    - Классы векторных типов данных в ICC
    - Встроенные атрибуты векторных типов в GCC
- Директивы компилятора
- Векторизуемые операции с массивами
- Векторизующий компилятор
- Библиотеки векторизованных подпрограмм

Средства векторизации
Вставки на ассемблере (микрокодирование)

Где работает:
- Работает на всех компиляторах, допускающих ассемблерные вставки
- Встроенный ассемблер должен знать используемые команды

Пример: сложение двух 4-элементных векторов
с использованием расширения SSE

```
typedef struct{
        float x, y, z, w;
} Vector4;

void SSE_Add(Vector4 *res, Vector4 *a, Vector4 *b){
        asm volatile ("mov %0, %%eax"::"m"(a));
        asm volatile ("mov %0, %%ebx"::"m"(b));
        asm volatile ("movups (%eax), %xmm0");
        asm volatile ("movups (%ebx), %xmm1");
        asm volatile ("addps %xmm1, %xmm0");
        asm volatile ("mov %0, %%eax"::"m"(res));
        asm volatile ("movups %xmm0, (%eax)");
}
```

Средства векторизации
Векторные операции и типы данных в языке

Встроенные в компилятор операции (intrinsics) и типы данных
- Для каждого представления векторного регистра есть свой тип данных
- Для каждой векторной команды процессора есть своя встроенная функция

Где работает:
- На большинстве известных компиляторов (gcc, clang, icc, cl.exe, …)
- Компилятор должен поддерживать используемое векторное расширение

Пример: скалярное произведение векторов длины n, кратной 4-м,
с использованием расширения SSE

```
#include <xmmintrin.h>
float inner(int n, float* x, float* y){
  __m128 *xx = (__m128*)x;
  __m128 *yy = (__m128*)y;
  __m128 s = _mm_setzero_ps();
  for(int i=0; i<n/4; ++i){
    __m128 p = _mm_mul_ps(xx[i],yy[i]);
    s = _mm_add_ps(s,p);
  }
 __m128 p = _mm_movehl_ps(p,s);
  s = _mm_add_ps(s,p);
  p = _mm_shuffle_ps(s,s,1);
  s = _mm_add_ss(s,p);
  float sum;
  _mm_store_ss(&sum,s);
  return sum;
}
```

Intel Intrinsics Guide:
https://software.intel.com/sites/landingpage/IntrinsicsGuide/

Средства векторизации
Векторные операции и типы данных в языке

Классы векторных типов данных в Intel C++ Compiler

- Для каждого представления векторного регистра есть свой класс
- Для каждой векторной команды процессора есть свой метод
  - обёртка над SIMD intrinsics
- Дополнительные методы и операторы для работы с векторами
    - add_horizontal, mul_horizontal, flip_sign, length, length_sqr, dot, normalize, <<, [], …

Где работает:

- Intel C++ Compiler, необходимо подключить ivec.h / fvec.h / dvec.h

Пример: скалярное произведение векторов длины n, кратной 4-м,
с использованием расширения SSE

```
#include<fvec.h>

float inner(int n, float* x, float* y){
  F32vec4 *xx = (F32vec4*)x;
  F32vec4 *yy = (F32vec4*)y;
  F32vec4 s; s.set_zero();
  for(int i=0; i<n/4; ++i)
    s += xx[i] * yy[i];
  return add_horizontal(s);
}
```

Средства векторизации
Векторные операции и типы данных в языке

Встроенные атрибуты векторных типов в GCC

- Векторные типы данных: __attribute__((vector_size(16)))
- Перегруженные обычные операции: +, *, >=, >>, …
- Встроенные операции: __builtin_shuffle(a,b,mask)

Где работает:

- gcc, clang

Пример: вычисление квадрата разности двух 4-элементных векторов

```
typedef float v4f __attribute__ ((vector_size (16)));

float inner(int n, float* x, float* y){
  v4f *xx = (v4f*)x;
  v4f *yy = (v4f*)y;
  v4f s = {0.0f, 0.0f, 0.0f, 0.0f};
  for(int i=0; i<n/4; ++i)
    s += xx[i] * yy[i];
  return s[0] + s[1] + s[2] + s[3];
}
```

GCC vector extension:
https://gcc.gnu.org/onlinedocs/gcc/Vector-Extensions.html

Средства векторизации
Директивы компилятора

- Директивы распараллеливания циклов на основе OpenMP
- Программист сам следит за корректностью применения директив

Где работает:

- Intel C/C++ Compiler (#pragma simd)
- Компиляторы, поддерживающие OpenMP 4.0
  - icc, gcc-4.9 –fopenmp-simd

- Пример: скалярное произведение векторов длины n:

```
float inner(int n, float* x, float* y){
  float s = 0.0f;
  #pragma omp simd reduction(+:s)
  for(int i=0; i<n; ++i)
    s += x[i] * y[i];
  return s;
}
```

Intel SIMD vectorization:
https://software.intel.com/ru-ru/node/512635
OpenMP 4.0:
http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf

Средства векторизации
Векторизуемые операции с массивами

- Правила выделения секций массивов
- Скалярные операции и функции определены для массивов
- Функции для редукции по элементам массива

Где работает Cilk Plus:

- icc, gcc-4.9 -fcilkplus

Пример:

Fortran 90	Расширение Intel Cilk Plus для C/C++

`real*8 a(N,N), b(N,N), c(N,N)`
`a(1:N/2,:) = -1.0`
`a(N/2+1:N,:) =  1.0`
`b = 2.0`
`c = sin(a) + b*5.0`
	`double a[N][N], b[N][N], c[N][N];`
	`a[  0:N/2] = -1.0;`
	`a[N/2:N/2] =  1.0;`
	`b[:][:] = 2.0;`
	`c[:][:] = sin(a[:][:]) + b[:][:]*5.0;`

Пример: скалярное произведение векторов длины n:

```
float inner(int n, float x[n], float y[n]){
  return __sec_reduce_add(x[:] * y[:]);
}
```

Intel Cilk Plus:
https://www.cilkplus.org/

Средства векторизации

Векторизующий компилятор

- Компилятор распознаёт циклы, которые могут быть векторизованы, и векторизует их
- Пользователь может сообщать компилятору дополнительную информацию и пожелания с помощью директив
- Где работает:
  - gcc (циклы попроще), icc (циклы посложнее)
- Пример: скалярное произведение векторов длины n:

```
float inner(int n, float* x, float* y){
  float s = 0.0f;
  for(int i=0; i<n; ++i)
    s += x[i] * y[i];
  return s;
}

$icc –vec-report=3 test.c
…
test.c(21): (col. 3) remark: LOOP WAS VECTORIZED
…
```

Intel automatic vectorization:
https://software.intel.com/ru-ru/node/512629

Средства векторизации

Библиотеки векторизованных подпрограмм

- Библиотека подпрограмм, которые уже реализованы с использованием векторных расширений
- Пример: операция вычисления скалярного произведения векторов из библиотеки BLAS MKL

Где работает: везде

```
#include<mkl_blas.h>

float inner(int n, float* x, float* y){
  int inc = 1;
  return SDOT(&n,x,&inc,y,&inc);
}
```

ATLAS: http://math-atlas.sourceforge.net/
Intel MKL: https://software.intel.com/en-us/intel-mkl
AMD ACML: http://developer.amd.com/tools-and-sdks/cpu-development/amd-core-math-library-acml/

АВТОМАТИЧЕСКАЯ ВЕКТОРИЗАЦИЯ В INTEL C/C++ COMPILER

Автоматическая векторизация в Intel C/C++ Compiler

- Ограничения на автоматическую векторизацию
  - Рассматриваются только самые внутренние циклы
  - Цикл должен правильной структуры: for (i=0;i<N;i++)
    - Границы и шаг цикла – инварианты (не меняются внутри или снаружи)
    - Тело цикла не должно иметь других точек входа и выхода (return, break, continue, goto, …)
  - Тело цикла не должно быть слишком сложным
  - Итерации цикла должны быть независимыми на дистанции размера вектора (компилятор может генерировать несколько версий кода)
  - Типы данных должны быть векторизуемыми
  - Вызываемые функции должны иметь векторизованные варианты
  - Векторизация цикла должна быть выгодна

Зависимости по данным

- Flow (True) dependence – Read After Write
  - X := 10 — нельзя устранить
  - Y := X + C
- Anti dependence – Write After Read
  - X := Y + C — можно устранить
  - Y := 10
- Output dependence – Write After Write
  - X := 10 — можно устранить
  - X := 20
- Input dependence – Read After Read
  - Y := X + 3 — не нужно устранять
  - Z := X + 5

Автоматическая векторизация в Intel C/C++ Compiler

- Распараллеливание возможно, если нет цикловых зависимостей между итерациями
- Векторизация возможна, если нет цикловых зависимостей между каждым выражением в теле цикла с глубиной меньшей или равной длине вектора

Пример 1:	Пример 2:

`for (i=0; i<N; i++)`
`{`
`  y[i] = a*x[i] + b;`
`}`
	`for (i=5; i<N; i++)`
	`{`
	`  y[i] = a*x[i] + b*y[i-5];`
	`}`

Возможна векторизация и распараллеливание	Возможна векторизация с длиной вектора не больше 4

Автоматическая векторизация в Intel C/C++ Compiler

- Когда код векторизуется?
  1. Компилятор доказал отсутствие цикловых зависимостей, используя только доказанные факты о программе.
  2. Компилятор доказал отсутствие цикловых зависимостей, сделав худшие предположения там, где был недостаток доказанных фактов.
  3. Компилятор считает, что векторизация будет выгодной.

Автоматическая векторизация в Intel C/C++ Compiler

- Когда код векторизуется?
  1. Компилятор доказал отсутствие цикловых зависимостей, используя только доказанные факты о программе.
  2. Компилятор доказал отсутствие цикловых зависимостей, сделав худшие предположения там, где был недостаток доказанных фактов.
  3. ~~Компилятор считает, что векторизация будет выгодной.~~

```
#pragma vector always
```

Считать, что векторизация всегда выгодна

Автоматическая векторизация в Intel C/C++ Compiler

- Когда код векторизуется?

1. Компилятор доказал отсутствие цикловых зависимостей, используя только доказанные факты о программе.
2. ~~Компилятор доказал отсутствие цикловых зависимостей, сделав худшие предположения там, где был недостаток доказанных фактов.~~
3. Компилятор считает, что векторизация будет выгодной.

```
#pragma ivdep
```

Считать, что скрытых зависимостей нет

Автоматическая векторизация в Intel C/C++ Compiler

- Когда код векторизуется?

1. ~~Компилятор доказал отсутствие цикловых зависимостей, используя только доказанные факты о программе.~~
2. ~~Компилятор доказал отсутствие цикловых зависимостей, сделав худшие предположения там, где был недостаток доказанных фактов.~~
3. ~~Компилятор считает, что векторизация будет выгодной.~~

`#pragma simd / #pragma omp simd`

Векторизовать в любом случае,
даже если это испортит программу

Автоматическая векторизация в Intel C/C++ Compiler

- Как посмотреть:
  - векторизовался ли код?
  - почему не векторизовался?
- Сообщения компилятора:
  - Ключ: icc –vec-report=3
- Посмотреть ассемблерный листинг
  - Ключ: icc –S

Автоматическая векторизация в Intel C/C++ Compiler

- Пример:

```
void vsum(int n, double *x, double *y, double *z)
{ int i;

  for (i=0;i<n;i++) z[i] = x[i] + y[i];
}
```

- Компилятор сгенерирует несколько версий кода:
  - Скалярную для случая, если указатели пересекаются
  - Векторную для случая, если указатели не пересекаются

Автоматическая векторизация в Intel C/C++ Compiler

- Пример:

```
void vsum(int n, double *x, double *y, double *z)
{ int i;
  #pragma ivdep
  for (i=0;i<n;i++) z[i] = x[i] + y[i];
}
```

- Компилятор сгенерирует векторный код

Автоматическая векторизация в Intel C/C++ Compiler

- Пример:

```
double add(double a, double b) { return a+b; }

void vsum(int n, double *x, double *y, double *z)
{
  int i;
  for (i=0;i<n;i++) z[i] = add(x[i],y[i]);
}
```

Автоматическая векторизация в Intel C/C++ Compiler

- Пример:

```
__attribute__((vector))
double add(double a, double b) { return a+b; }

void vsum(int n, double *x, double *y, double *z)
{ int i;
  #pragma ivdep
  for (i=0;i<n;i++) z[i] = add(x[i],y[i]);
}
```

Автоматическая векторизация в Intel C/C++ Compiler

- Пример: в нотации Intel C/C++ Compiler

```
__attribute__((vector))
double add(double a, double b) { return a+b; }

void vsum(int n, double *x, double *y, double *z)
{ int i;
  #pragma simd
  for (i=0;i<n;i++) z[i] = add(x[i],y[i]);
}
```

- Пример: в нотации OpenMP 4.0

```
#pragma omp declare simd
double add(double a, double b) { return a+b; }

void vsum(int n, double *x, double *y, double *z)
{ int i;
  #pragma omp simd
  for (i=0;i<n;i++) z[i] = add(x[i],y[i]);
}
```

Автоматическая векторизация в Intel C/C++ Compiler

- Пример: в нотации OpenMP 4.0

```
#pragma omp declare simd uniform(c)
double add(double a, double b, double c)
{ return a + b + c; }

void vsum(int n, double *x, double *y, double *z)
{ int i;
  #pragma omp simd
  for (i=0;i<n;i++) z[i] = add(x[i],y[i],0.5);
}
```

NEC
SX-9