

ПРОБЛЕМЫ СОЗДАНИЯ ЭФФЕКТИВНОГО ПАРАЛЛЕЛЬНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Зав.каф. АДМ мехмата ЮФУ,
д.т.н., Штейнберг Б.Я.

Зачем нужны быстрые вычисления



Где применяются быстрые вычисления?

- ▶ В управлении сложными объектами (АЭС, самолеты,...)
- ▶ В обороне (раннее обнаружение противника,...)
- ▶ В сетевых серверах (в т.ч. ИНТЕРНЕТ)
- ▶ В экономических прогнозах
- ▶ В банковских расчетах
- ▶ В криптографии
- ▶ В научных исследованиях
- ▶ В искусственном интеллекте (эра роботов в Японии уже наступила!)

КОМПЬЮТЕРЫ И СУПЕРКОМПЬЮТЕРЫ

- Ранее высокопроизводительные вычисления относились только к суперкомпьютерам.
- Суперкомпьютеры производились в единичных экземплярах и почти всегда были синонимом параллельных компьютеров
- Сейчас параллельные компьютеры у всех на столах, а высокопроизводительные вычисления находят все более широкие применения.

Проблемы эффективности последовательных программ



КОМПЬЮТЕРЫ МЕНЯЮТСЯ БЫСТРЕЕ, ЧЕМ ПРОГРАММЫ

- ▶ Мы привыкли к преемственности в смене IBM-совместимых компьютеров (с системой команд X86): новые компьютеры работают быстрее и на них переносятся старые программы.
- ▶ Но так ли эффективно эти программы переносятся?

Изменились условия оптимизации программ.

Новые архитектуры требуют новых оптимизирующих преобразований программ

- ▶ $X(i+2) = X(i+2)+5$
- ▶ заменять следующим кодом?
- ▶ $K = i+2$
- ▶ $X(k) = X(k)+5$
- ▶ Замена уменьшает количество операций, но вводит новую переменную k . Если для этой переменной не окажется свободного регистра, то придется обращаться к памяти и существенно потерять быстродействие.
- ▶ Такая замена давала ускорение на старых процессорах, использовалась в старых пакетах прикладных программ и однозначно давала ускорение.
- ▶ 30 лет назад время доступа к данным было быстрее времени обработки, а сейчас – наоборот и во много раз!

Следует осмотрительно использовать старые библиотеки программ.

- Для задачи перемножения матриц $N \times N$ есть стандартный алгоритм (N^3 умножений и $2 * N^3$ чтений данных) и алгоритм Штрассена, ($N^{2.8}$ умножений и $15 * N^{2.8}$ чтений данных).
- В разные годы численные эксперименты (каф. АДМ РГУ) определяли, начиная с какой размерности матриц N алгоритм Штрассена эффективнее:
 - 1988 г. $N=32$
 - 2000 г. $N=512$
- В 2008 г. должно быть $N=25000$, нет компьютера для такого эксперимента.

Проблемы эффективности параллельных программ



Разным задачам – разные архитектуры

- ▶ Циклы с условными операторами выгодно отображать на асинхронные архитектуры
- ▶ Самые глубоко вложенные циклы, если итерации их независимы – на векторную архитектуру, а если зависимы – на конвейер.

ПРОБЛЕМЫ СОЗДАНИЯ ЭФФЕКТИВНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

- Нет универсальных архитектур. Разные задачи эффективны на разных архитектурах
- Разработка параллельных программ требует высокой квалификации программистов, больше времени, мер обеспечения надежности.
- Переписывать низкоуровневые программы

ДОЛГО И ДОРОГО.

ПУТИ РАЗВИТИЯ ИНДУСТРИИ ЭФФЕКТИВНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

- Создание систем полуавтоматических преобразований программ (распараллеливающих и оптимизирующих), как инструмента в разработке эффективного ПО
- Создание систем высокоуровневой переносимости ПО на основе систем преобразования программ
- Системы типа «сначала программа - затем компьютер» на основе систем преобразования программ
- Модификация подготовки программистов, ориентированная на РАЗЛИЧНЫЕ виды параллельного программирования.
- Модификация теории сложности вычислений, учитывающая обращения к памяти
- Модификация теории преобразования программ, ориентированная на оптимизацию обращений к памяти
- Создание математического аппарата для автоматической работы с общей распределенной памятью.

Исследовательские университетские распараллеливающие системы

- POLARIS – распараллеливающая система Urbana University (штат Illinois), руководитель проекта David Padua.
- SUIF –Stanford University Intermediate Format - система Стэнфордского университета.
- PIP – Parametric Integer Programming – система анализа программ P. Feautrier (Фр.)
- PARAWISE
- *Исследования перерастают в коммерческие проекты*

Южный федеральный университет мехмат

Открытая распараллеливающая система.

- ОРС – прототип инструмента создания эффективных параллельных программ
- В группе ОРС разрабатываются новые методы распараллеливания программ, методы автоматической генерации электронных схем.

ПАРАЛЛЕЛЬНО ВЫПОЛНЯЕМЫЕ ПРОГРАММНЫЕ ЦИКЛЫ.

- ▶ Под параллельным выполнением цикла понимается одновременное выполнение его итераций.
- ▶ *Какие итерации можно одновременно выполнять, а какие – нет, зависит от архитектуры параллельного компьютера и программных зависимостей.*

Определение распараллеливаемых циклов в ОРС. (помечены флажками слева)

В окнах информация о возможной векторизации или распараллеливании.
Для функций определяется отсутствие побочных эффектов.

The screenshot displays the Open Parallelizing System (OPS) v 3.0 interface. The main window shows a C program named `test1.c` with nested loops. The code is as follows:

```
min(a : ST_INT, b : ST_INT) : ST_INT
{
    a
    if (a > b)
    {
        return b
    }
    else
    {
        return a
    }
}
t(s : ST_INT) : ST_INT
{
    s
    for i = 1; (i <= 9); i = (i + 1)
    {
        for j = 1; (j <= 9); j = (j + 1)
        {
            for k = 1; (k <= 9); k = (k + 1)
            {
                U[i][j] = min(U[i][j], 1)
                for l = 1; (l <= 9); l = (l + 1)
                {
                    V[i][j] = (V[(i - 1) * (j - 1)] + 1)
                }
            }
        }
    }
    return 0
}
```

The code is annotated with flags on the left side of the editor. The `for` loops are marked with green flags, indicating they are candidates for parallelization. The `if` statement is marked with a red flag, indicating it is not parallelizable. The `return` statements are marked with blue flags, indicating they are not parallelizable.

The **Graphs** window shows a dependency graph for the code. The graph includes nodes for `U[i][j]` and `V[i][j]`, with edges representing data dependencies. A text box in the graph window states: "Потоковая зависимость Носители (уровни): 0".

The **sm - Блокнот** window displays the following text:

итерации цикла не могут выполняться независимо Цикл векторизуем.

The **Анализ программных циклов** window displays the following text:

SourceMarker наглядно отображает свойства циклов, полученные при помощи анализа проведенного средствами OPS.

Если рядом с циклом есть цветная метка, то после нажатия на неё на экране появится информация о цикле. Цвета метки отображают обнаруженные свойства.

Виды параллельных вычислений

- ▶ Параллельное выполнение задач
- ▶ Параллельное выполнение функций
- ▶ *Параллельное выполнение циклов*
- ▶ Параллельное выполнение операций
- ▶ Параллельное выполнение микрокоманд

Распараллеливание циклов

- ▶ Под параллельным вычислением цикла понимается одновременное выполнение его итераций.
- ▶ Термин «одновременное» допускает разные толкования и требует дополнительных пояснений.
- ▶ Распараллеливанию циклов могут мешать программные зависимости.

Граф информационных связей (Dependence graph)

- ▶ Вершины графа – вхождения переменных.
- ▶ Дуга соединяет пару вершин $(v1, v2)$ тогда и только тогда, когда оба вхождения $v1$ и $v2$ обращаются к одной и той же ячейке памяти (зависимы), причем $v1$ раньше, а $v2$ позже.

Пример графа информационных связей, автоматически построенного в ОРС

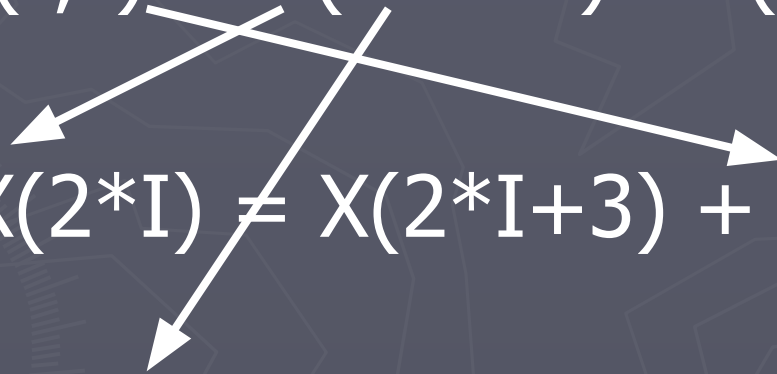
DO 99 J=2,N

DO 99 I=2,N

A(I,J) = X(2*I+2) - B(I,J)

X(2*I) = X(2*I+3) + A(I,J-1)

99 X(2*I+2) = B(I,J+2)

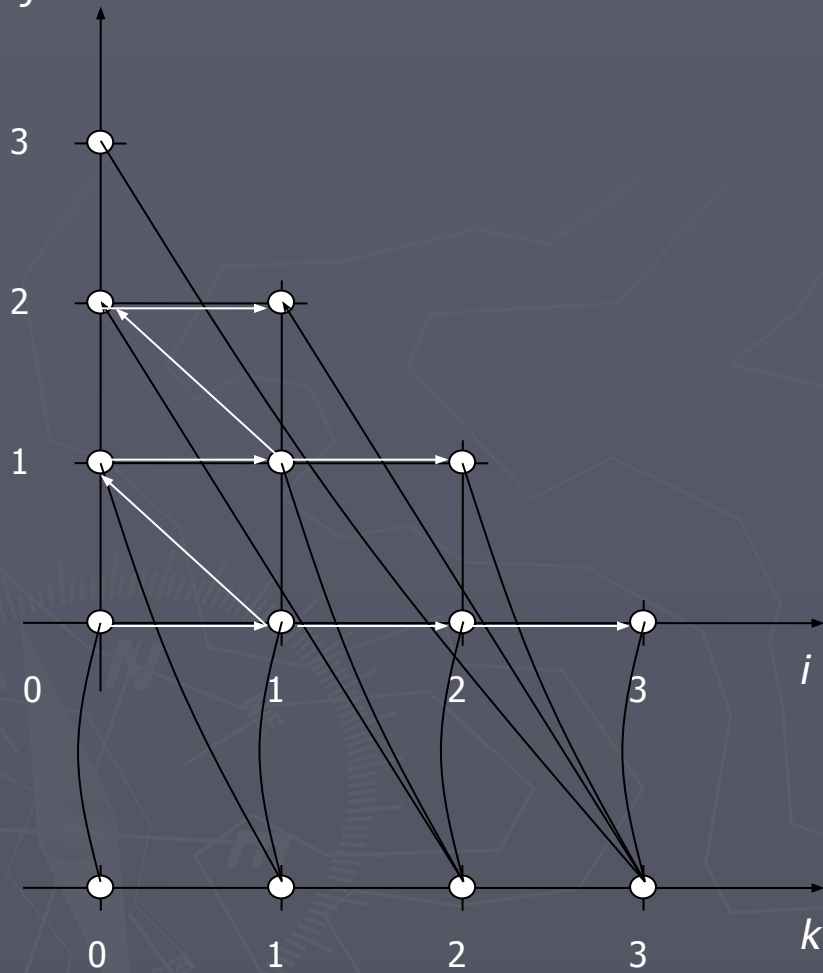


Циклически независимая и циклически порожденная зависимости. Пример.

- ▶ DO 99 I=2,N
- ▶ B(I) = A
- ▶ v1 v2
- ▶ 99 A = B(I) + B(I-3)
- ▶ v3 v4 v5

- ▶ v1 -> v4, v2 -> v3, v3 -> v2 циклически независимые зависимости
- ▶ v1 -> v5 циклически порожденная зависимость
- ▶ v1 -> v1 циклически независимая самозависимость

Пример (А.М. Шульженко). фрагмент программы умножения двух многочленов и решетчатый граф:



```
For (k=0; k<=(N+M); k++)  
    c[k]=0;  
%      u1  
    for (i=0; i<=M; i++)  
        for (j=0; j<=N; j++)  
            c[i+j]= c[i+j]+a[i]*b[j];  
%      u2      v1  v2  v3
```

История решетчатых графов

- Решетчатые графы использовались концептуально, для иллюстрации идей в методе гиперплоскостей Л. Лампорта, в методе параллелепипедов В. Вальковского и многих у других исследователей.
- Традиционные формы хранения графа в виде матрицы смежностей или в виде списка дуг не эффективны: прочтение такого графа может потребовать больше времени, чем последовательное исполнение программы, по которой этот граф построен.
- В конце 80-х годов был сделан прорыв в этом направлении: В.В. Воеводин и Р. Feautrier научились хранить этот граф в виде функций, и разработали алгоритмы построения этих функций. У Р. Feautrier функция строится для программ из линейного класса и содержит в своем определении не очень удобные для исследования условные операторы. У В.В. Воеводина функция является кусочно-линейной и определена на подмножестве практически значимых программ линейного класса.

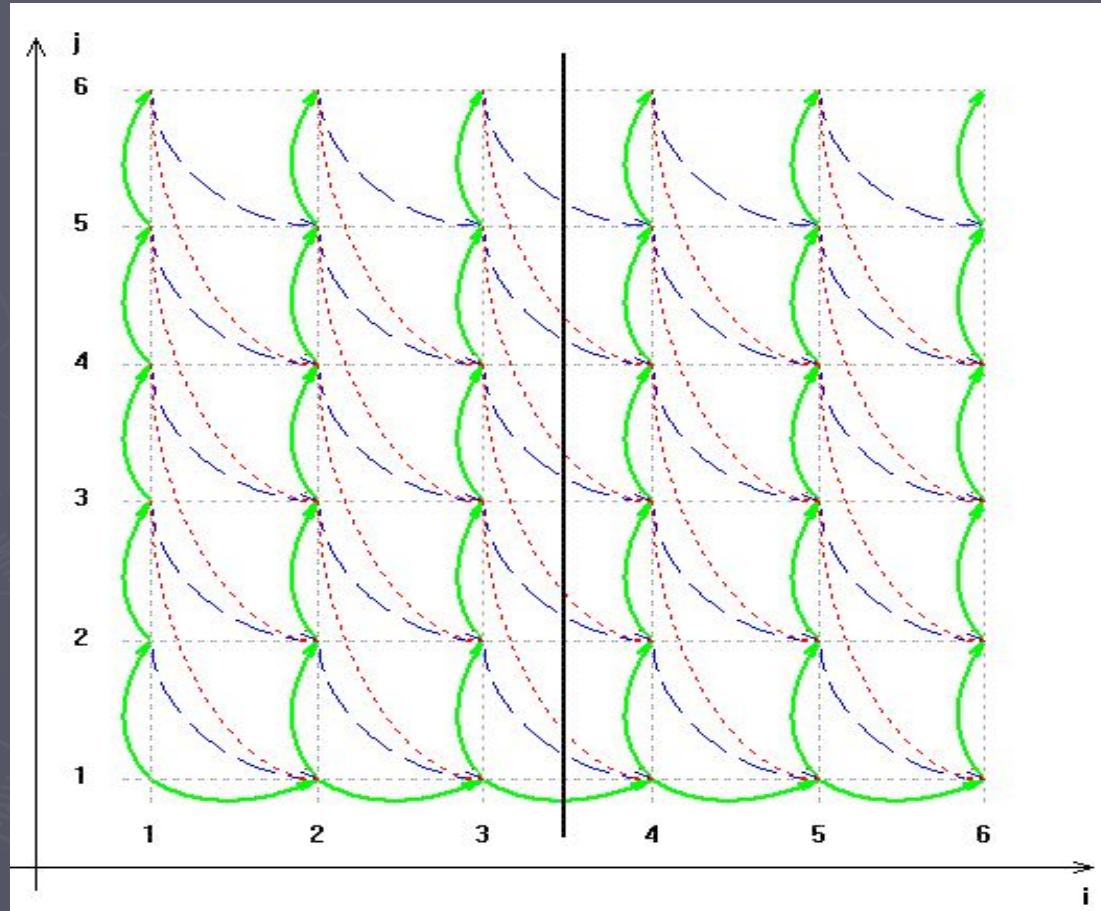
Направления работ группы ОРС по использованию решетчатых графов в распараллеливающих компиляторах:

- ▶ Использование решетчатых графов в преобразованиях программ, которые не могут быть основаны на традиционном графе программных зависимостей.
- ▶ Использование решетчатых графов для отображения программ на параллельную архитектуру.
- ▶ Расширение класса программ, к которому применимы методы теории решетчатых графов.

Пример анализа зависимостей с использованием решетчатого графа

- ▶ Пример.
- ▶ DO 99 i = 1, N
- ▶ DO 99 j = 1, M
- ▶ 99 $X(i+j) = A(i,j)*X(i+j-1)+B(i,j)$

Полный решетчатый граф программы с дугами истинной зависимости, антивисимости и выходной самозависимости (граф построен с помощью ОРС).



- При разбиении программы на два потока барьеры нужны на каждой итерации цикла со счетчиком j .

Преобразованная программа со вспомогательными массивами и только одним барьером.

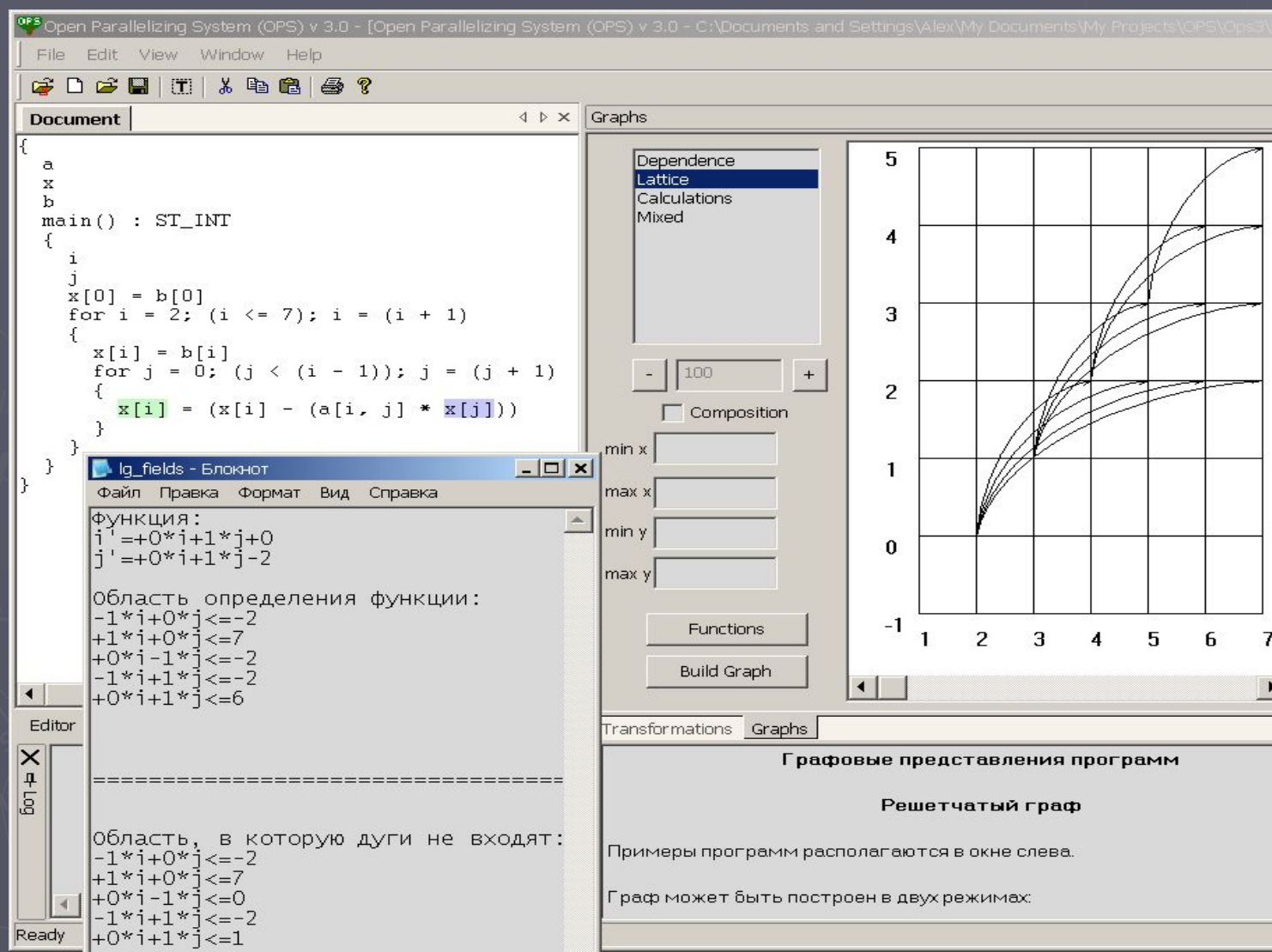
```
➤ For (i=0; i <= N; ++i) {
➤   a[i+1] = ... ;
➤   ... = a[i];
➤ }
➤ //----- барьер -----
➤ For (i=1; i <= N/2; ++i) {
➤   b[1] = a[i];
➤   for(j=2; j <= N; ++j) {
➤     b[j] = ... ;
➤     ... = b[j-1];
➤   }
➤ }
➤ //===== фрагменты 3 и 4 выполняются независимо =====
➤ For (i=N/2+1; i <= N; ++i) {
➤   c[1] = a[i];
➤   for (j=2; j <= N; ++j) {
➤     c[j] = ... ;
➤     ... = c[j-1];
➤   }
➤ }
➤ For (j=2; j <= N; ++j)
➤   a[N+j] = c[j];
```

(2)

(3)

(4)

Визуализация элементарного решетчатого графа для выделенной пары вхождений переменной в ОРС. В специальном окне представлены функции, описывающие решетчатый граф.



3D-визуализация решетчатого графа в OPS.

graph4 - Программа просмотра изображений и факсов

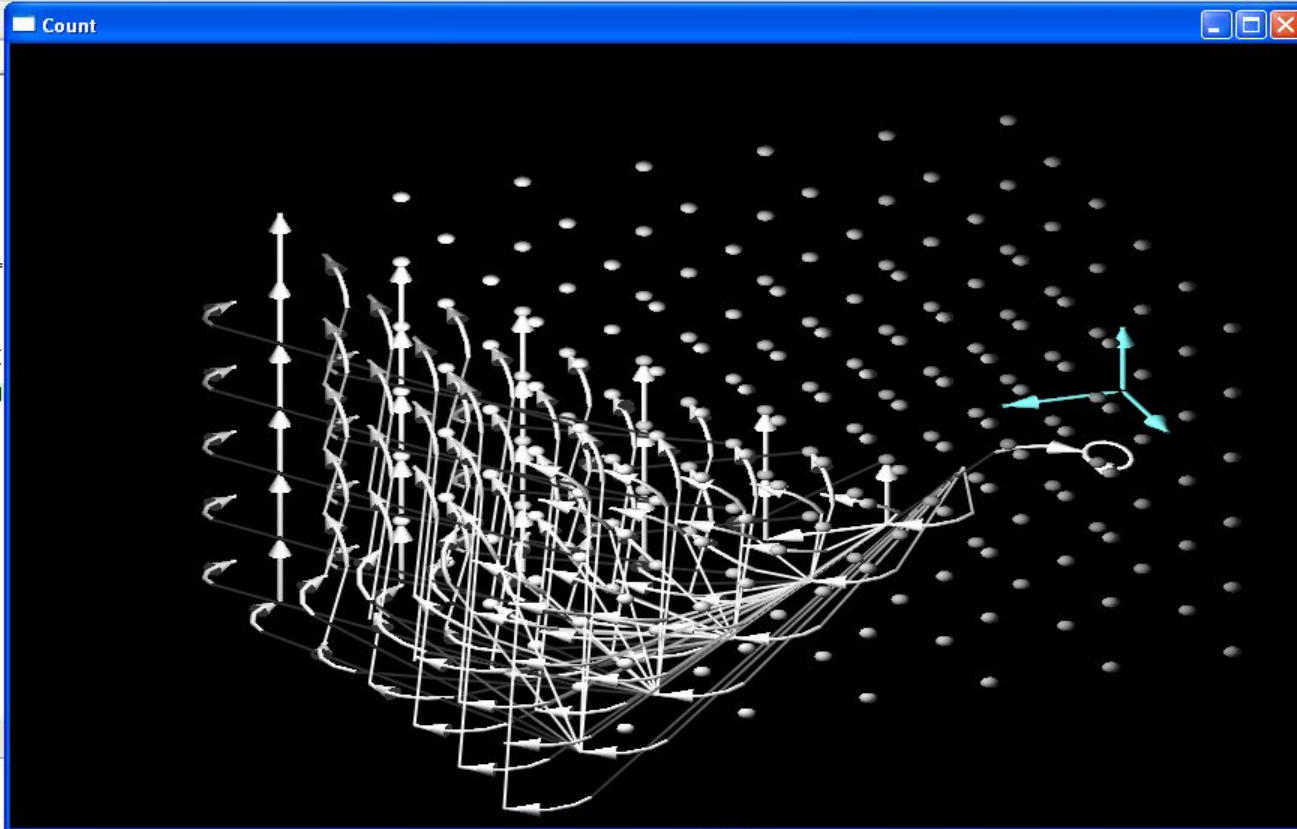
OPS Open Parallelizing System (OPS) v 3.0 - [C:\OPSA\Ops3\bin\samples\Graph Lattice\slae_revsubst_2.c]

File Edit View Window Help

Count

Document

```
{
a
x
b
main() : ST_INT
{
i
j
k
x[0] = b[0]
for i = 1; (i <=
{
x[i] = b[i]
for j = 0; (j
{
for k = 0; (
{
x[(i + k)]
}
}
}
}
```



Editor Selector Marker

Х
д
б
д
б

Примеры программ располагаются в окне слева.

Граф может быть построен в двух режимах:

1. Установлен флажок "композиция графов". Строится композиция решетчатых графов для всех пар

Ready

пуск OPS Open Parallelizing Sys... Count

EN 18:01

Открытая распараллеливающая система. Исследования.

- Автоматические оценки погрешностей
- Распараллеливание рекуррентных циклов
- Максимальное разбиение программных циклов
- Подстановка и переименование индексных переменных
- Оптимизация регистровых сбросов
- Размещение данных в параллельной памяти
- Многоконвейерные вычисления
- Оптимизирующий/распараллеливающий конвертор с языка Си на язык описания электронных схем VHDL

Открытая распараллеливающая система (ОРС)

- ОРС позволяет автоматически строить граф информационных связей решетчатый граф программы. В ОРС автоматически определяются параллельно выполняемые циклы.
- Руководитель: д.т.н. Штейнберг Б.Я.
- Состав группы: студенты, магистранты, аспиранты, сотрудники нескольких кафедр мехмата Южного федерального университета.



Открытая
распараллеливающая
система

<http://ops.rsu.ru>