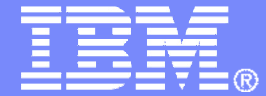


IBM HRL

Data Layout Optimizations in GCC

Olga Golovanevsky Razya Ladelsky

IBM Haifa Research Lab, Israel

© 2007 IBM Corporation

Agenda

- **General idea**
- **In scope of GCC**
- **Structure reorganization optimizations**
  - Transformations
  - Decision making mechanism
  - Type–escape analysis
- **Matrix reorganization optimizations**
  - Matrix flattening
  - Matrix transposing
  - Combining: flattening + transposing
- **Current Status**
- **Experimental results**

© 2007 IBM Corporation

General Idea

- **General idea: to adapt the layout of a data structure to its access patterns**
- **Principle: if data elements are accessed close in time, they should be allocated close in memory to gain better cache utilization**
- **Optimizations - profile-based/static:**
  - matrix flattening, matrix transposing
  - structure peeling, splitting and reordering
- **Flow:**
  - analyze data accesses
  - change allocations, deallocations and access sites
- **For structure:**
  - fields are manipulated
- **For matrix:**
  - dimensions are manipulated

 © 2007 IBM Corporation

In scope of GCC

- **Presented:**
  - matrix flattening/transposing:
    - gcc summit 2006
  - struct-reorg:
    - gcc summit 2005 by M.Hagog and C.Tice
    - gcc summit 2007
- **Based on tree-ssa IR**
- **Use ipa infrastructure**

  Jan Hubicka/Suse-Novell
- **Require whole program**
  - -fwhole-program, -combine
- **Potential for LTO**
- **Developed on:**
  - matrix optimizations – ipa-branch
  - struct-reorg – struct-reorg-branch
- **Scheduled for gcc4.3 mainline**

© 2007 IBM Corporation

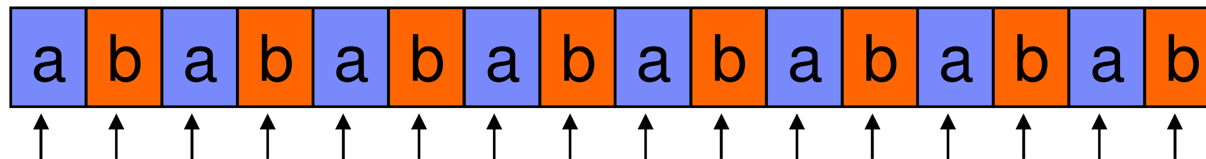
Structure peeling

typedef struct {int a; float b;} str;

str *arr = (str *) malloc(N*sizeof(str));

```
foo1 ()
{
  for (i=0; i<N; i++)
    arr[i].a = …;
}
```

```
foo2 ()
{
  for (i=0; i<N; i++)
    … = arr[i].b;
}
```



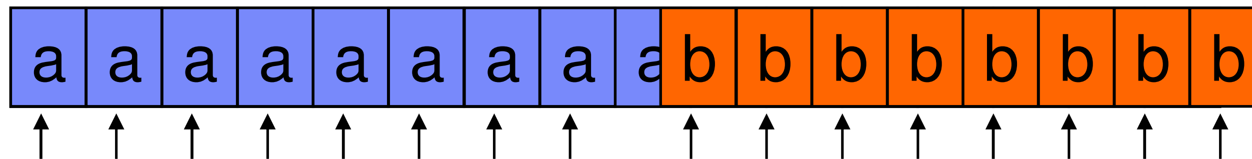
 © 2007 IBM Corporation

Structure peeling

```
typedef struct {int a; float b;} str;
typedef struct {int a;} str1;
typedef struct {float b;} str2;

str *arr = (str *) malloc(N*sizeof(str));
str1 *arr1 = (str1 *) malloc(N*sizeof(str1));
str2 *arr2 = (str2 *) malloc(N*sizeof(str2));
```

```
foo1 ()                          foo2 ()
{                                {
  for (i=0; i<N; i++)              for (i=0; i<N; i++)
    arr[i].a = ...;                  ... = arr[i].b;
    arr1[i].a = ...;                 ... = arr2[i].b;
}                                }
```

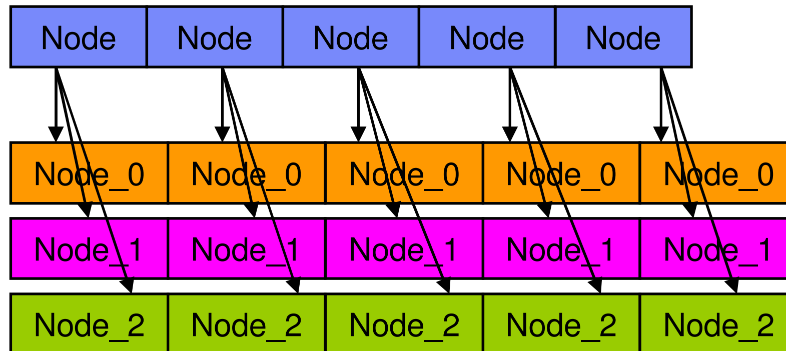


© 2007 IBM Corporation

Structure Splitting

```
typeset struct node_struct  {
    struct node_struct *parent;
    Expression      left;
    Operator *      bin_op;
    Expression      right;
} Node;
```

```
typedef struct node_struct  {
            Node_0 * Node_0_ptr;
            Node_1 * Node_1_ptr;
            Node_2 * Node_2_ptr;
} Node;
```




```
typedef struct  {
    Expression      right; } Node_0;
typedef struct  {
    Expression      left;
    Operator *      bin_op; } Node_1;
typedef struct  {
    struct node_struct *parent; } Node_2;
```


© 2007 IBM Corporation

Structure Reordering

```
typedef struct Operator_struct
{
   OpType       type;
   n_char       symbol;
   const n_char *name;
   OpMode       mode;
   n_int        precedence;
   struct Operator_struct *    higher;
   struct Operator_struct *    lower;
} Operator;
```

```
typedef struct Operator_struct
{
   n_int            precedence;
   OpType           type;
   OpMode           mode;
   n_char           symbol;
   struct Operator_struct  *lower;
   const char*      name;
   struct Operator_struct  *higher;
} Operator;
```

- Less aggressive optimization
- Does not change allocation/access sites, only type definition

© 2007 IBM Corporation

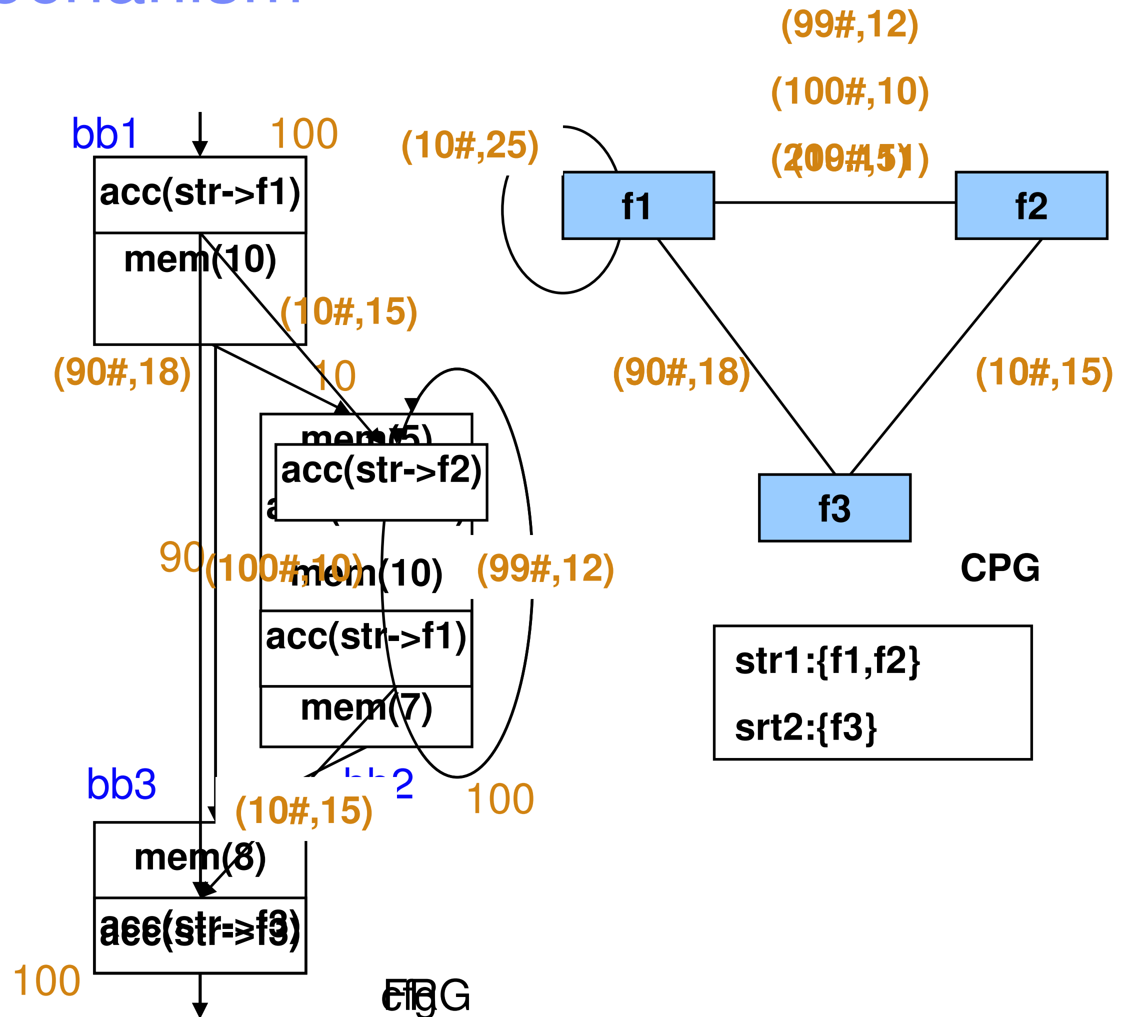
Decision Making Mechanism

- **profile-based; make use of CFG**
- **Fields Reference Graph:**
  - vertices – field accesses
  - edges
    - follow CFG
    - weighted with amount of data accessed between two field accesses
- **Rebuild Fields Reference Graph into Close Proximity Graph**
  - vertices – fields
  - Edges
    - potentially all vertices are interconnected
    - average amount of data accessed between two fields

© 2007 IBM Corporation

Decision Making Mechanism

```
struct str {int f1; int f2; int f3};
foo ()
{
   …
   acc(str->f1);
   mem(10);
   for (i=0; i<N; i++)
   {
      mem(5);
      acc(str->f2);
      mem(10);
      acc(str->f1)
      mem(7);
   }
   mem(8);
   acc (str->f3);
   …
}
```



© 2007 IBM Corporation

Type escape analysis

- **structure escapes when:**
  - pointer to structure is parameter/return type of global function
  - structure is global variable
  - one of the fields is escaped
  - cast to pointer to structure
  - operations other than plus, minus or times are applied to pointer to structure

- **ipa-type-escape**
  - ipa pass, by Kenneth Zadeck
  - initially written on gimple
  - recently updated to tree-ssa
  - by default with –O2

© 2007 IBM Corporation

Future plans

- **Support transformation for substructure**
- **Analyze/reorg more that one structure at a time:**
  - consider the hierarchy of structures as a whole unite
  - mixed data structures: arrays within structures
- **Spec2000/2006/mcf benchmark**
  - new type of transformation
- **Analyze more data, tune decision make algorithm**
- **Type escape analysis**
  - more extensions with tree-ssa
  - check real world applications with big portion of the code is in libraries

© 2007 IBM Corporation

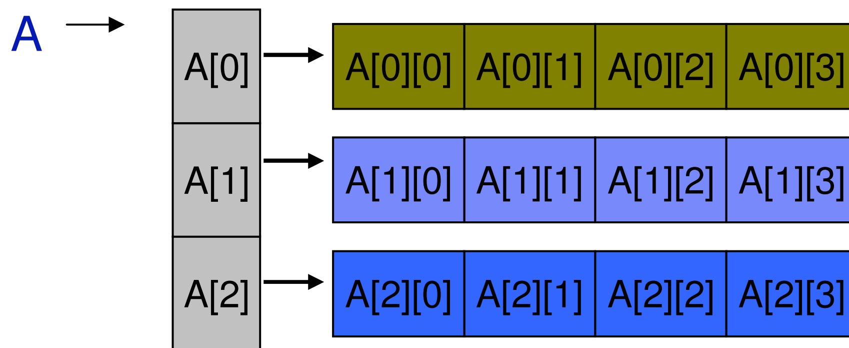
Agenda

- **General idea**
- **In scope of GCC**
- **Structure reorganization optimizations**
  - Transformations
  - Decision making mechanism
  - Type–escape analysis
- **Matrix reorganization optimizations**
  - Matrix flattening
  - Matrix transposing
  - Combining: flattening + transposing
- **Current status**
- **Experimental results**

© 2007 IBM Corporation

Matrix Flattening

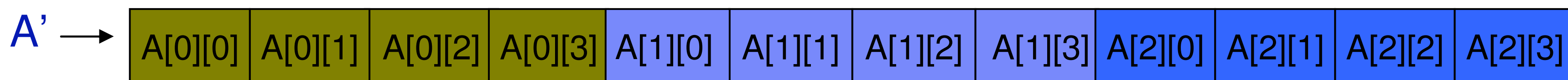
Before flattening:



Access A[i][j] :

1. base = A + i * size(ptr)
2. A [i] [j] = *(*base + j * size(el))

After flattening:



One memory allocation

Access A[i][j] : A [i] [j] = *(A' + (i * 4 + j) * size(el))

© 2007 IBM Corporation

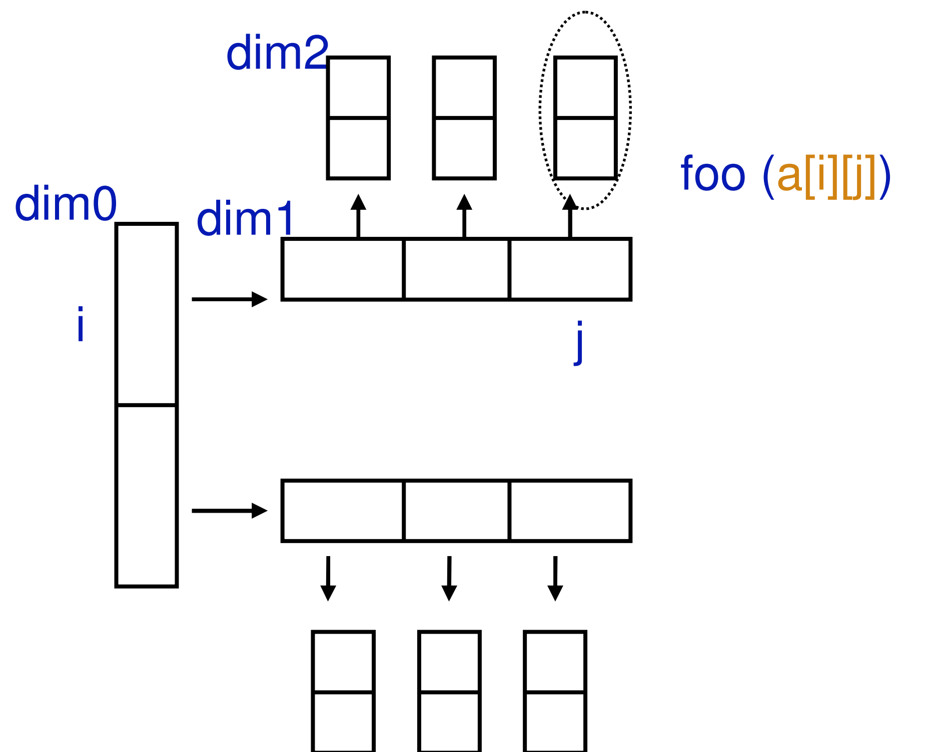
Flattening - the benefit

- **What is the benefit?**
  - Each matrix access involves less indirections -> less memory references
  - The rows are laid one after the other -> contiguous space allocation -> increase cache locality

 © 2007 IBM Corporation

2.1 Matrix flattening: partial flattening

Flattening dim0 and dim1



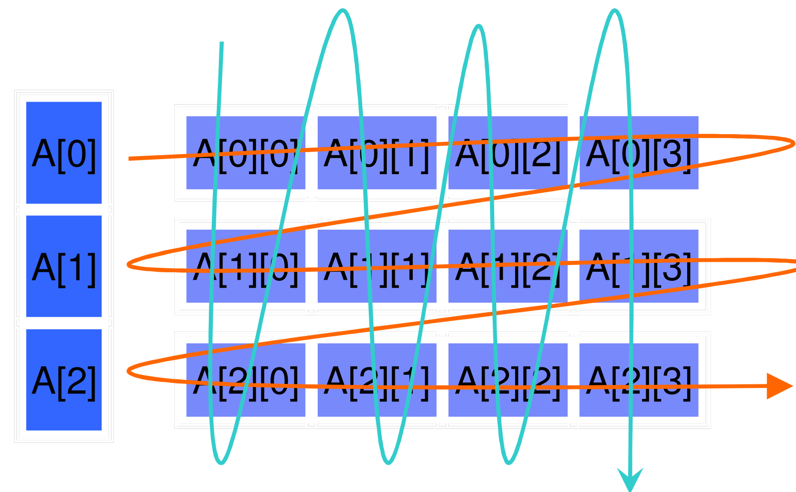
flattened

foo (a[i][j])

 © 2007 IBM Corporation

Matrix Transposing

```
for (i=0; i<N; i++)
   for (j=0; j<M; j++)
      access to  A[ j ][ j ]
```



The elements of the columns rows are accessed sequentially

Iterated according to the columns rows

Matrix transposing: Interchanging the dimensions

© 2007 IBM Corporation

Matrix Transposing: decision making - example

```
    for (i=0; i<N; i++)        -> row major matrix
      for (j=0; j<M; j++)
acc1:     access to A[ i ][ j ]

    for (i=0; i<N; i++)        -> column major matrix
      for (j=0; j<M; j++)
acc2:     access to A[ j ][ i ]
```


© 2007 IBM Corporation

Matrix Transposing: decision making

- Multi dimensional ->

  all possible permutations for allocation

  - How to decide the best permutation?

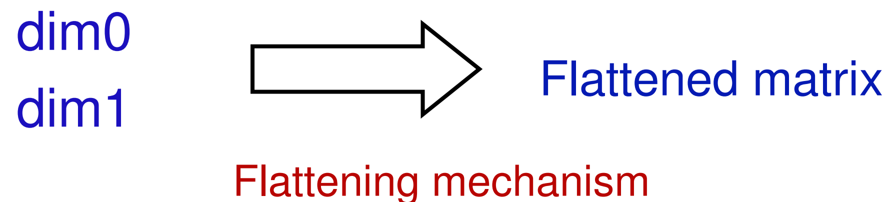
  - We consider:
    - Which dimensions are iterated in the innermost loops
    - Profiling information: how often the accesses were executed

  - The hotness of dimension determines how close in time were accesses to the elements of this dimension, normalized by the hotness of access execution.

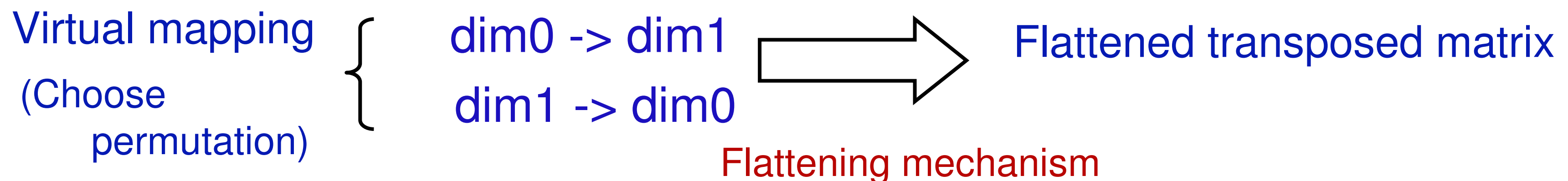
© 2007 IBM Corporation

Combining Flattening and Transposing

- **Original matrix :**



- **Transposed matrix :**



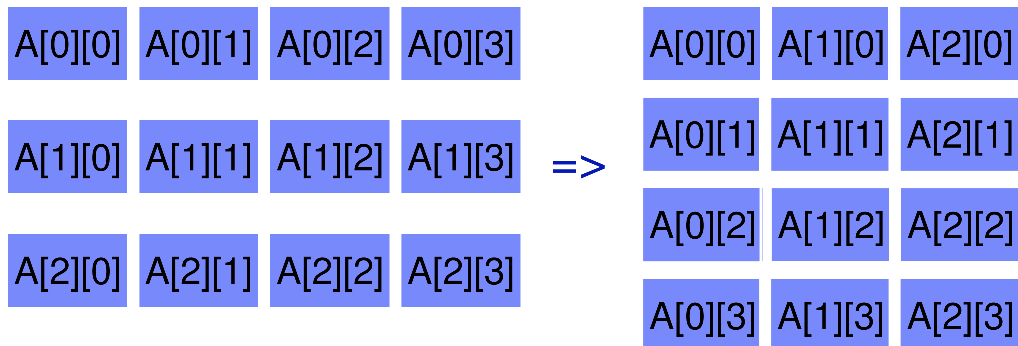
© 2007 IBM Corporation

Combining Flattening and Transposing

transposing + flattening ->

flatten the transposed matrix

A[3,4] => A[4,3]



© 2007 IBM Corporation

status

Matrix flattening and transposing:

- Initially developed on the ipa branch.
- To be included in GCC4.3; patch submitted recently
- Enabled by -fipa-mreorg, -fdump-ipa-mreorg for dump
- Transposing enabled when profiling information is available

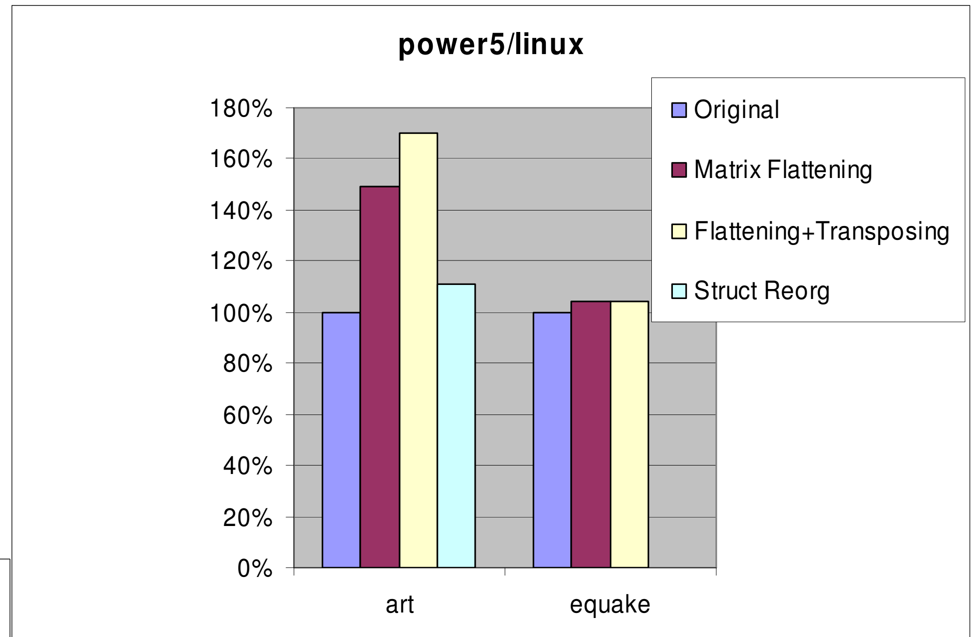
Struct reorg:

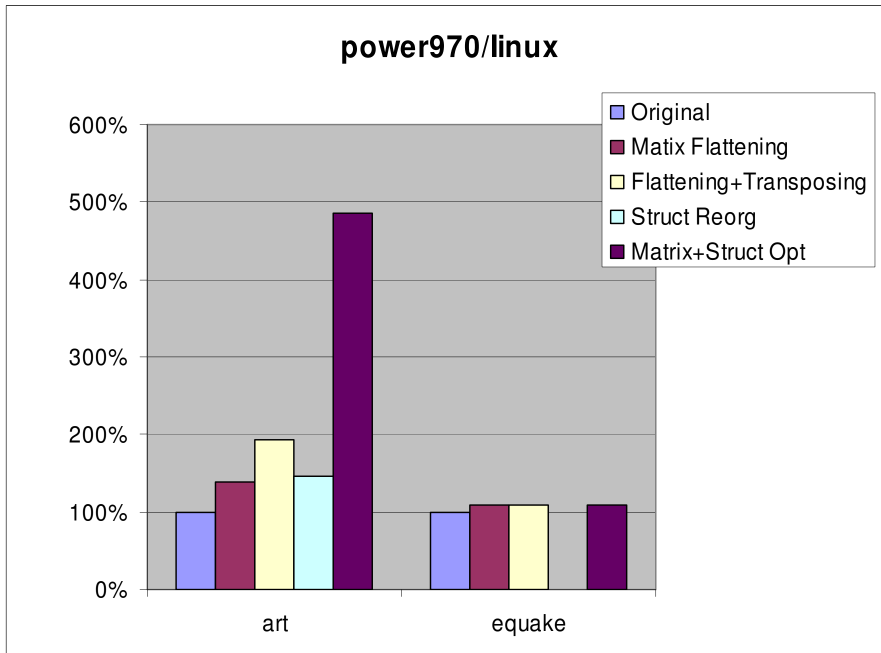
- Initially developed on struct reorg branch
- To be included in GCC4.3
- Enabled by -fipa-reorg-structs

Thanks! Mustafa Hagog, Revital Eres, Daniel Berlin, Kenneth Zadeck, Zdenek Dvorak, Jan Hubicka, Sebastian Pop

© 2007 IBM Corporation

Performance Results





© 2007 IBM Corporation

Questions?

© 2007 IBM Corporation