Using the AIMMS Sparse Execution Engine

Chris Kuip
AIMMS Client Support

Webinar, May 28, 2014

Presentation sections

- Introduction Sparse Execution with example
- Glimpse under the hood
- Tackling AIMMS performance challenges

Why sparse execution engine?

1. Purpose: support algebraic notation
    1. Use identifiers of high dimension P_{ijtps}
    2. Use indexes in large sets, `i`, `j` in `Nodes`, 1K–1M
2. Requires:
    1. Sparse storage – only store non-zero values
    2. Sparse execution – avoid computing value zero

Explicit to Implicit loops

For all i,j: $a_{i,j} = b_{i,j} + c_{i,j}$

```
For i Do
   For j Do
     a(i,j) := b(i,j) + c(i,j);
   EndFor ;
EndFor ;
```

```
a(i,j) := b(i,j) + c(i,j);
```

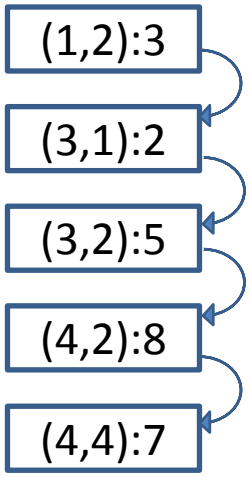
Implicit loops allow AIMMS to exploit sparseness.

Sparse storage, example: `A(i,j)`

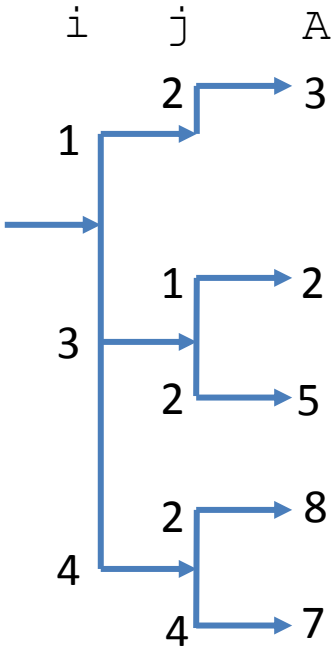
Stored full:

	1	2	3	4
1		3		
2				
3	2	5		
4		8		7

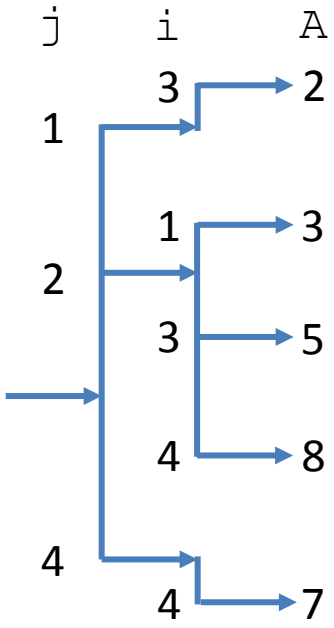
As list:



As index level tree:

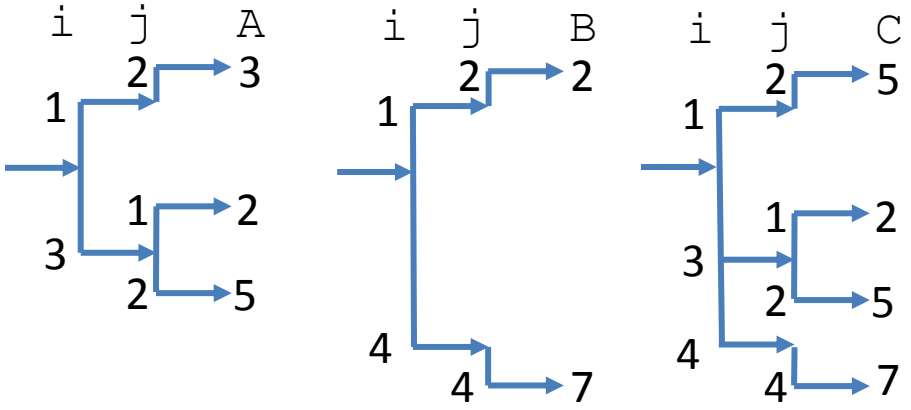


As permutation thereof:



Sparse execution: example +

```
c(i,j):=a(i,j)+b(i,j);
```

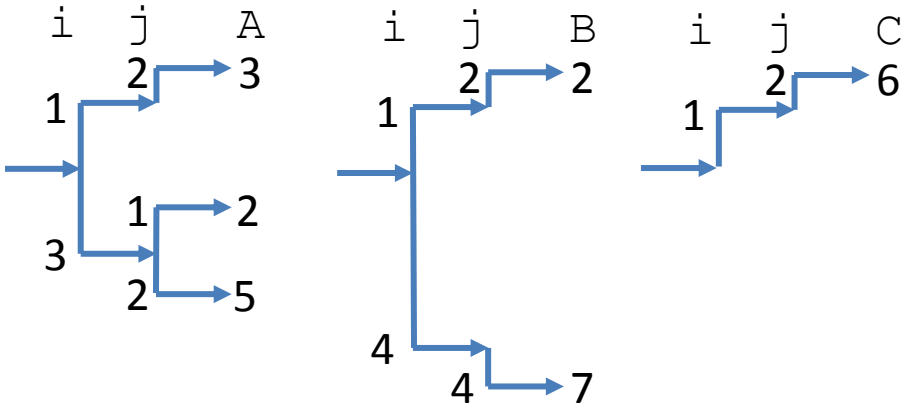


- Taking the **union** of sets of tuples.

Order of elements significant for proper matching.

Sparse execution: example *

```
c(i,j):=a(i,j)*b(i,j) ;
```



- Taking the **intersection** of sets of tuples.

Order of elements significant for searching.

Sparsity Methods for Unary and Binary Operators

Unary operator `u`:

Sparsity method	Behavior	Operator	For instance
Sparse	`u`(0)=0	-, sin	-0 → 0
Dense	`u`(0)<>0	not, cos	not 0 → 1

Binary operator `b`:

Sparsity method	Behavior	Operator	For instance
Intersection	`b`(0,x) → 0, `b`(x,0) → 0	*, AND, |	0*x → 0
Union	`b`(0,0) → 0	+, -, <	0+0 → 0
Dense	`b`(0,0) → <>0	^, /, <=	0<=0 → 1

Q: which sparsity method?

- For the operator "=" in

```
A(i,j) := 1|(b(i,j)=c(i,j));
```

1. Sparse
2. Intersection
3. Union
4. Dense

Webinar Polls are anonymous;
we only see number and average of votes!

A: which sparsity method?

- For the operator "**=**" in

```
A(i,j)  := 1|(b(i,j)=c(i,j));
```

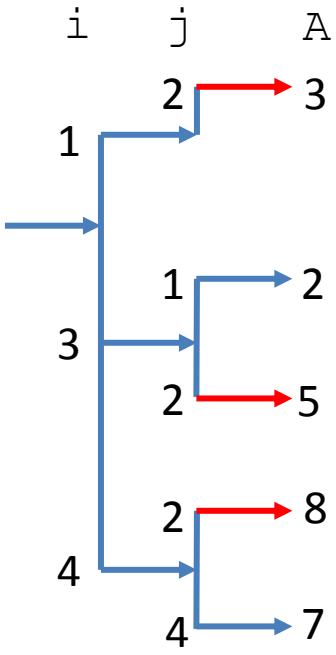
- Dense sparsity method; 0 = 0 results in a 1;
- The operation is executed for each combination of `i` and `j`.

Applying permutations:

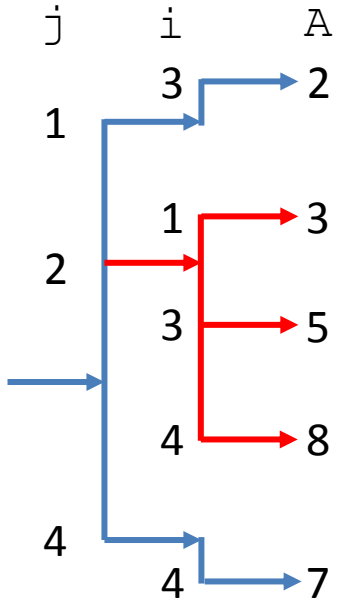
```
b(j):=sum(i,a(i,j))
```

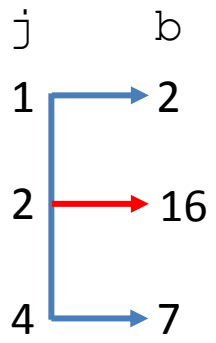
b(2)

As index level tree:



Uses permutation:





Q: Example, missing index:

```
A(i,j) := sum( k, b(k,i) );
```

- Can this be improved (if yes how)? 30 seconds
  1. No
  2. Yes, introduce a 1-dim identifier indexed over `i`
  3. Yes, use a FOR loop, carefully incrementing
  4. Yes, but needs complete rewrite

Webinar Polls are anonymous;
we only see number and average of votes!

A: Example, missing index:

```
A(i,j) := sum( k, b(k,i) );
```

- No dependence on `j`; improve by substitution:

```
AI(i) := sum( k, b(k,i) );
A(i,j) := AI(i) ;
```

We see this regularly in models with
long and complicated assignment statements, or
long and complicated constraints.

Fixed time overhead per executed statement

- Initialization:
  - Update definitions of referenced identifiers.
  - Determine per expression, the appropriate rule
- **Actually perform computation**
- Assign computed data to identifier(s)
- Internal administration (definition graph, modification time stamps)
- Work the GUI pages

Avoid execution of individual statements inside (nested) FOR loops.

Q: FOR loops 1, order apar.

Assignment: order the values in apar(i).

- With i index in intSet, a subset OrdIntSet, order by user, ep1 and ep2 element parameters in intSet, Consider the following FOR loop:

```
ordIntSet := sort( i, apar(i) );
for ( i ) do
        ep1 := i ;
        ep2 := element( ordIntSet, ord(i) );
        ordpar1(ep1) := apar(ep2);
endfor ;
```

- Can this be improved? 1 minute poll.
  1. No
  2. Yes, use a while loop
  3. Yes, add a condition on the for loop
  4. Yes, remove for loop, substitute element parameters

Webinar Polls are anonymous;
we only see number and average of votes!

A: FOR loops 1, order apar.

- With i index in intSet, a subset OrdIntSet, order by user, ep1 and ep2 element parameters in intSet. Consider the following FOR loop:

```
ordIntSet := sort( i, apar(i) );
for ( i ) do
        ep1 := i ;
        ep2 := element( ordIntSet, ord(i) );
        ordpar(ep1) := apar(ep2);
endfor ;
```

- First round, Remove for and add index i to parameters on lhs of :=.

```
ep1(i) := i; ! Becomes silly assignment, remove.
ep2(i) := element( ordIntSet, ord(i) );
ordpar(i) := apar(ep2(i));
```

- Substitute element parameter ep2(i) will give you:

```
ordpar(i):= apar(element( ordIntSet, ord(i) ));
```

- This is faster because the “fixed time overhead” per executed statement is avoided 3*card(intSet)-1 times.

Q: FOR loops 2, adapting bounds

- We need to ensure each lowerbound is less than the corresponding upper bound:

```
for (i,j) | UpperLimit(i,j) <= LowerLimit(i,j) do
        put FormatString("Upper limit for (%e,%e) lower than lower
   limit",i,j),/;
        LowerLimit(i,j) := UpperLimit(i,j);
endfor;
```

- What causes this FOR loop to be slow?
  1. Formulation error “<“ should be used
  2. “<=“ is a dense operator
  3. The condition is recomputed each iteration
  4. All of the above

Webinar Polls are anonymous;
we only see number and average of votes!

A: FOR loops 2, adapting bounds

- Lower the lowerbound, when it exceeds the upperbound:

```
for (i,j) | UpperLimit(i,j) <= LowerLimit(i,j) do
        put FormatString("Upper limit for (%e,%e) lower
  than lower limit",i,j),/;
        LowerLimit(i,j) := UpperLimit(i,j);
endfor;
```

- All, but most important: LowerLimit is adapted in each executed iteration, which makes AIMMS recompute the entire condition!

```
for (i,j) | UpperLimit(i,j) < LowerLimit(i,j) do
        put FormatString("Upper limit for (%e,%e) lower than
  lower limit",i,j),/;
endfor;
LowerLimit((i,j)| UpperLimit(i,j) < LowerLimit(i,j)) :=
          UpperLimit(i,j);
```

Q: FOR loops 3, stock over time.

- Assume `ProdStockDef(p,t)` has the following definition:

```
ProdStockDef(p,t-1) + Production(p,t) - Sales(p,t)
```

- Production is capped by both production and storage capacity:

```
For ( t ) do
    Production(p,t) := min[ProductionCap(p),
        maxStock(p)-ProdStockDef(p,t-1)+sales(p,t)];
Endfor ;
```

- What causes the performance glitch? 1 minute poll
  1. The use of the “min” operator; use if then else instead
  2. Interaction between defined parameter and updated parameter inside for loop
  3. There should be a condition on the for loop
  4. All of the above

Webinar Polls are anonymous;
we only see number and average of votes!

A: FOR loops 3, stock over time.

- Assume `ProdStockDef(p,t)` has the following definition:

```
ProdStockDef(p,t-1) + Production(p,t) - Sales(p,t)
```

- Production is capped by both production and storage capacity:

```
For ( t ) do
    Production(p,t) := min[ProductionCap(p),
        maxStock(p)-ProdStockDef(p,t-1)+sales(p,t)];
Endfor ;
```

2: Interaction between defined parameter and updated parameter inside for loop. You may want to substitute the definition:

```
For ( t ) do
    Production(p,t) := min[ProductionCap(p),
       maxStock(p)-ProductStock(p,t-1)+Sales(p,t)];
    ProductStock(p,t) := ProductStock(p,t-1) +
          Production(p,t) - Sales(p,t);
Endfor ;
```

A: Index ordering problem

```
A1(i,j,k,l,m):=B(i,l)*C(j,m)*D(k,l)*E(l,m)
    N N N 1 1

A2(i,l,m,j,k):=B(i,l)*C(j,m)*D(k,l)*E(l,m)
    N 1 1 1 1
```

`i,j,k,l,m` in set `S` with `N` elements.
`B,C,D,E` all `N` elements, linking one `i` to a `l`, etc.

Q: Index ordering problem, puzzle

- The order of indexes is irrelevant most of the times.
  Illustrating a notable exception:

```
a1(i,j):=sum(k,c1(j,k)*b1(i,k));
a2(i,j):=sum(k,c2(i,j)*b2(j,k));
```

(here sets are large, `b1`, `b2`, `c1` and `c2` sparse)

- Which one is computed faster?
  1. a1
  2. a2
  3. Approximately the same
  4. I do not know

Webinar Polls are anonymous;
we only see number and average of votes!

A: Index ordering problem, puzzle

- The order of indexes is irrelevant most of the times.
  Illustrating a notable exception:

```
a1(i,j):=sum(k,c1(j,k)*b1(i,k));
   N N        1
a2(i,j):=sum(k,c2(i,j)*b2(j,k));
   N 1        1
```

(here sets are large, `b1`, `b2`, `c1` and `c2` sparse)

Rewrite `a1` as:

```
t(i,k,j):=c1(j,k)*b1(i,k); ! Select i, selects k.
                           ! Select k, selects j.
                           ! No trials needed.
a1(i,j) := sum(k,t(i,k,j));
  ! AIMMS' permutations make this quick.
```

References

- AIMMS Language Reference, Part IV: Sparse Execution.
- AIMMS Tech blog: http://blog.aimms.com/2011/12/tricks-to-improve-aimms-execution-time/

A moment of reflection

- Please reflect briefly on this webinar and answer:

- Are you now more confident tackling bottlenecks?
  - No, but I do not think I need it
  - No, I need more training
  - Mmmh, I am going to try it myself first
  - Yes, bring it on

Webinar Polls are anonymous;
we only see number and average of votes!

Announcement of next webinar

- The next webinar in this series, titled "*Analyzing infeasible problems in AIMMS*", will be presented by Marcel Hunting, Senior AIMMS Developer.

- Join us – Wednesday June 25, 2014:
  - 10 AM CET and
  - 8AM PDT/11AM EDT

- Register at http://business.aimms.com/news-events/product-training/