# The Locality Principle

Peter J. Denning
Presented by: Matthew Perez

# Background History

- Atlas Computer system (1949)
  - Storage: RAM and Disk
- Manual page transfers (overlays)
  - Programmers could utilize RAM and Disk
- Innovation: Virtual Memory (1959)
  - OS managed pages loaded on RAM


Atlas Computer System

# Challenges with Virtual Memory

- Issue 1: Translating addresses to locations

- Issue 2: Replacing loaded pages

# Issues with Virtual Memory

- Issue 1: Translating addresses to locations
  - Solution: High-speed cache (TLB)

- Issue 2: Replacing loaded pages

# Issues with Virtual Memory

- Issue 1: Translating addresses to locations
  – Solution: High-speed cache (TLB)

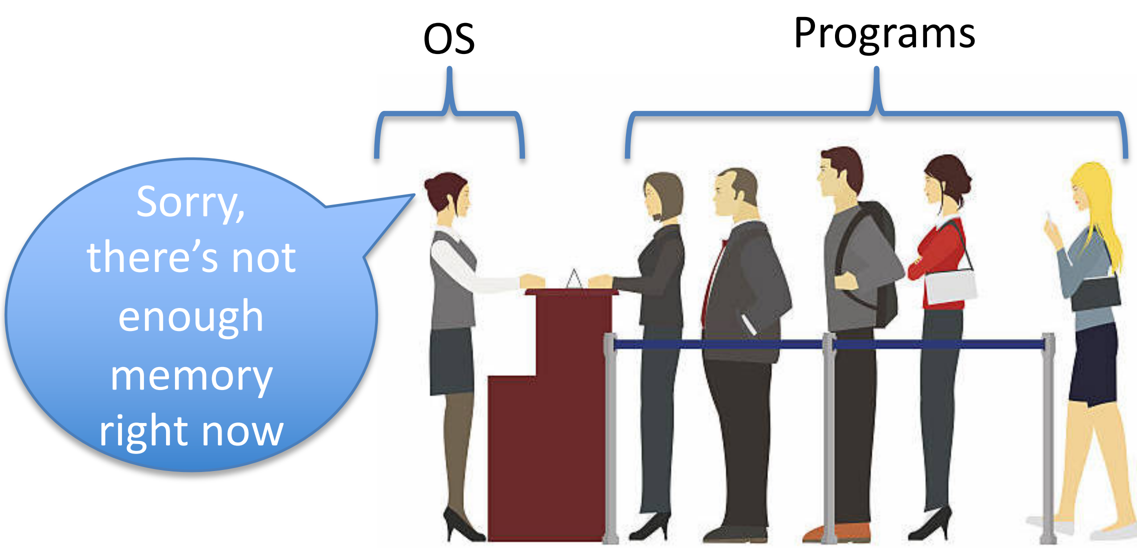
- Issue 2: Replacing loaded pages
  – Solution: Learning algorithms

# Page Faults

- Page faults -> 10,000x slower than CPU instruction cycle
- Industry invested in replacement algorithms
  - Bad algorithm = expensive in long run
  - IBM Watson found LRU to be best (1966)

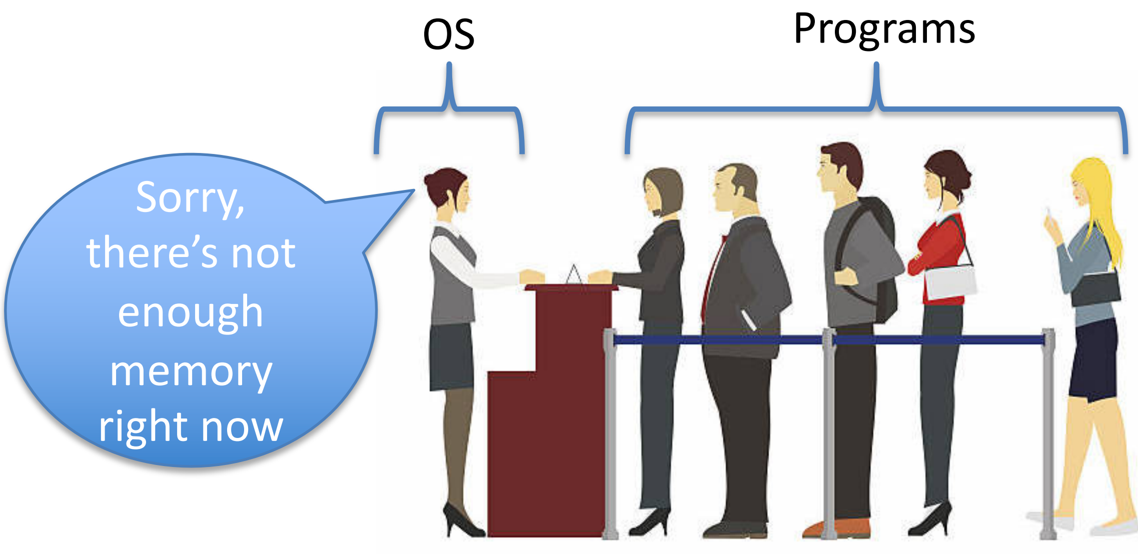
# Thrashing Problem

- Multiprogramming -> thrashing
- Too many programs = page faults
- Working set = Memory needed for program
- Solution: Limit multiprogramming



# Open Question: Thrashing

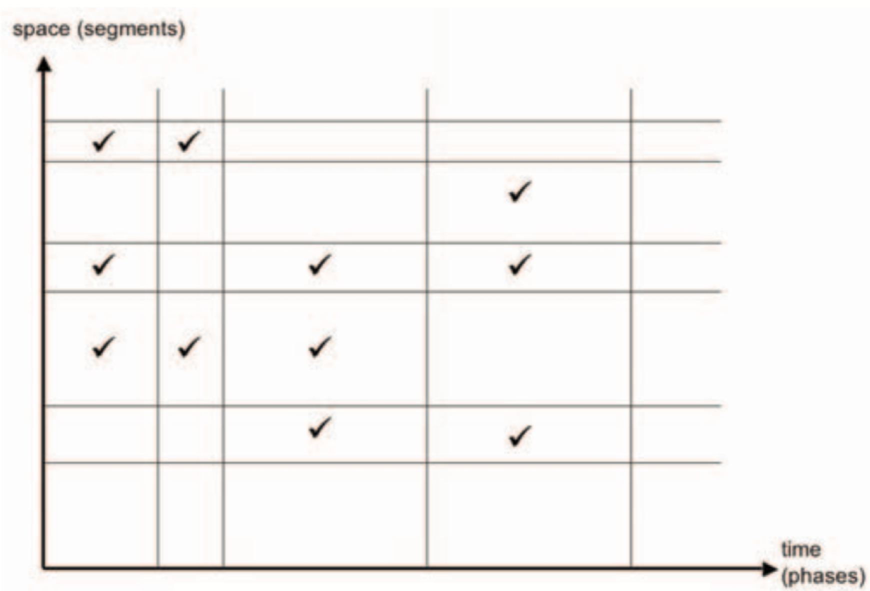
- How should we process programs in queue?
  - Let smaller programs go first?
  - How to ensure big programs get processed?



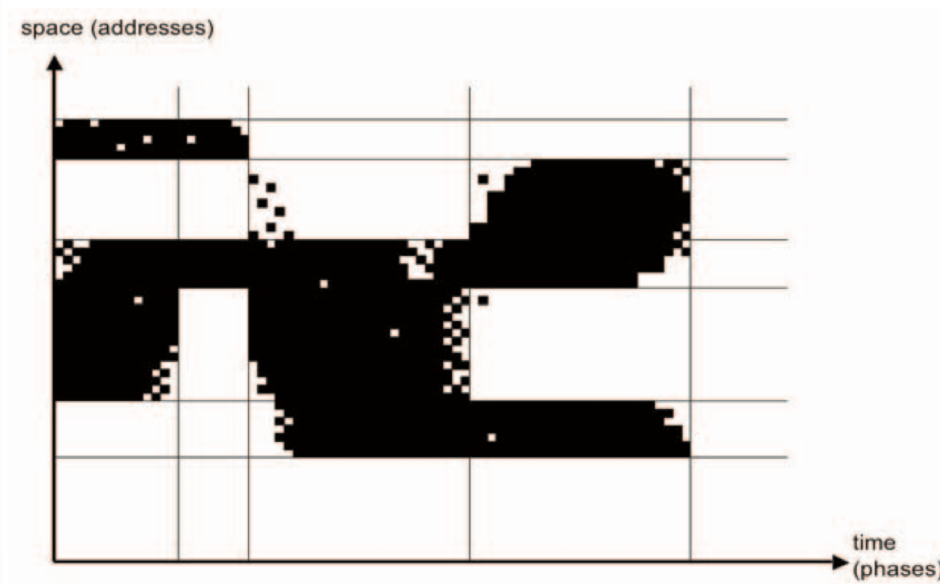
# Locality Principle

- Memory demand represents sequence of locality sets
- Programs reference addresses close in proximity and time
  - Spatial Locality = Nearby addresses
  - Temporal Locality = Recently referenced
- Beneficial for quickly loading memory

# Locality Visualization




Pre-planned




Non-planned

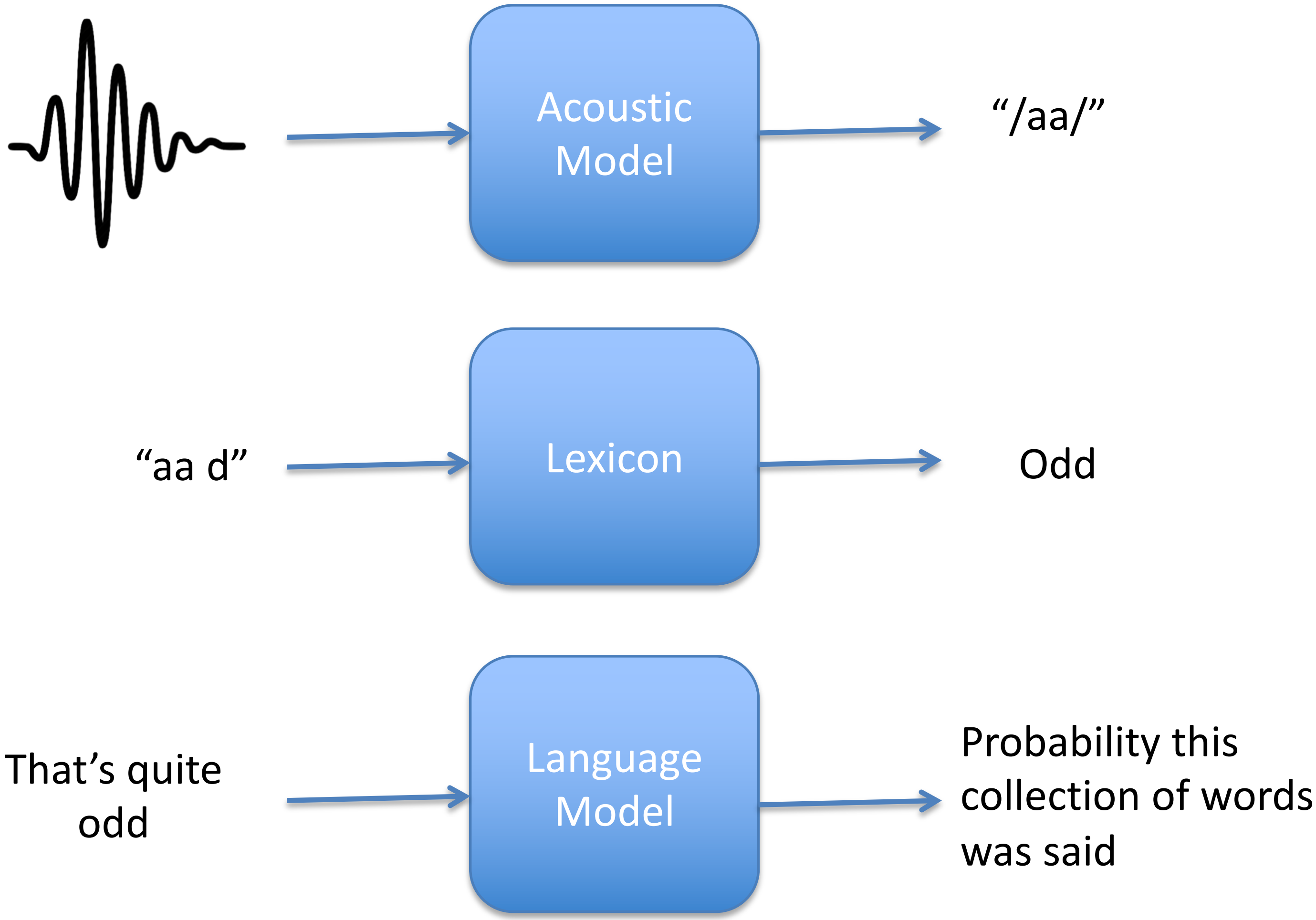
# Locality Principle Examples

- Loading data caches
- Web browsers – recent web pages
- Video streaming – reduce distance to server
- Search engines – relevant queries

# Personal Example

- Use locality principle to infer context?
- Speech recognition uses context for improving classification

# Speech Recognition



# ASR Example

- Just Acoustic Model and Lexicon (no context):

"He pored the waiter into a cap"

- Causes:
  - High speaker variability (accents, gender, word selection)
  - Noisy environment

# ASR Example

- Acoustic Model and Lexicon produce:

"He poured the ___ into a cup"

- Top acoustic model and lexicon choices:
  - Waiter
  - Otter
  - Water

Language Model decides most probable choice

# ASR Example

- Acoustic Model and Lexicon produce:

"He poured the ___ into a cup"

- Top acoustic model and lexicon choices:
  - Waiter
  - Otter
  - Water

# Discussion Questions

- Strengths
- Weaknesses
- Open Questions

# Discussion Questions

- Contrast with Exokernel. Virtual Memory gives OS more control rather than programmer (overlays). What are some of the pros/cons behind this approach?

- How to schedule/manage programs in the stall queue?

# Discussion Questions

- Analyze replacement algorithms: Random, FIFO, LRU, empty block. Which is best for small block sizes? Large block sizes?
- What about dynamically allocating different sized memory?

# Compiler Optimizations for Improving Data Locality

Steve Carr Kathryn S. McKinley Chau-Wen Tseng

Presented by: Wenjia He
Department: CSE

# Motivation

- Fast processor & slow memory
- Cache: based on data locality
- Before:

  Scientific programmers' effort

  Machine-dependent programming
- Now:

  Compiler strategy — reducing number of cache line references in loop nests

# Motivation

## Example — Matrix Multiply

Given N*N matrices A and B, calculate C = A*B

Loop nest:

```
Do J = 1, N
    Do K = 1, N
        Do I = 1, N
            C(I, J) = C(I, J) + A(I, K) * B(K, J)
```

# Background

- Data reuse
  - Temporal reuse
  - Spatial reuse
- Reference group
  - Group-temporal reuse
  - Group-spatial reuse

# Background

- Cost model

$$\text{RefCost} = \begin{cases} 1 & \text{loop invariant} \\ \text{trip}/(\text{cls}/\text{stride}) & \text{consecutive} \\ \text{trip} & \text{non-consecutive} \end{cases}$$

$$\text{LoopCost} = \sum_{k=1}^{m} \text{RefCost}_k \prod_{i \neq l} \text{trip}_i$$

# Key Insights and Design Details

- Compound loop transformations
  - Loop permutation
  - Loop reversal
  - Loop fusion
  - Loop distribution

# Key Insights and Design Details

- Compound loop transformations
  - Loop permutation
  - Loop reversal
  - Loop fusion
  - Loop distribution

# Key Insights and Design Details

- Loop permutation

  Steps:
  - Calculate LoopCost for each loop
  - Rank loops in order
  - Or find the nearest legal order

  Complexity
  - Sorting: O(nlogn)
  - Finding legal order: $O(n^2)$

# Key Insights and Design Details

- Loop permutation

  Example

```
Do J = 1, N
    Do K = 1, N
        Do I = 1, N
            C(I, J) = C(I, J) + A(I, K) * B(K, J)
```

# Key Insights and Design Details

- Loop permutation

  Example

| References | J | K | I |
|---|---|---|---|
| **C(I, J)** | $n*n^2$ | $1*n^2$ | $1/4*n*n^2$ |
| **A(I, K)** | $1*n^2$ | $n*n^2$ | $1/4*n*n^2$ |
| **B(K, J)** | $n*n^2$ | $1/4*n*n^2$ | $1*n^2$ |
| **Total** | $2n^3 + n^2$ | $5/4*n^3 + n^2$ | $1/2*n^3 + n^2$ |

# Key Insights and Design Details

- Compound loop transformations
  - Loop permutation
  - Loop reversal
  - Loop fusion
  - Loop distribution

# Key Insights and Design Details

- Loop reversal
  - Reverse the order of iteration
  - May help loop permutation

# Key Insights and Design Details

- Compound loop transformations
  - Loop permutation
  - Loop reversal
  - Loop fusion
  - Loop distribution

# Key Insights and Design Details

- Loop fusion
  - Combine multiple loop nests to one loop nest
  - Benefit
    - Create a permutable loop nest
    - Improve temporal locality

# Key Insights and Design Details

- Loop fusion

```
Example
Do I = 2, N
    Do K = 1, N
        X(I, K) = X(I-1, K) * A(I, K) / B(I-1, K)
    Do K = 1, N
        B(I, K) = A(I, K) * A(I, K) / B(I-1, K)
```

# Key Insights and Design Details

- Loop fusion

  Example

```
Do I = 2, N
    Do K = 1, N
        X(I, K) = X(I-1, K) * A(I, K) / B(I-1, K)
        B(I, K) = A(I, K) * A(I, K) / B(I-1, K)
```

# Key Insights and Design Details

- Loop fusion

```
Example
Do K = 1, N
    Do I = 2, N
        X(I, K) = X(I-1, K) * A(I, K) / B(I-1, K)
        B(I, K) = A(I, K) * A(I, K) / B(I-1, K)
```

# Key Insights and Design Details

- Compound loop transformations
  - Loop permutation
  - Loop reversal
  - Loop fusion
  - Loop distribution

# Key Insights and Design Details

- Loop distribution
  - Divide a single loop into multiple loops
  - Benefit: create permutable loop nests

# Key Insights and Design Details

- Loop distribution

  Example: Cholesky factorization

```
Do K = 1, N
    A(K, K) = SQRT(A(K, K))
    Do I = K+1, N
        A(I, K) = A(I, K) / A(K, K)
        Do J = K+1, I
            A(I, J) = A(I, J) - A(I, K) * A(J, K)
```

# Key Insights and Design Details

- Loop distribution

  Example: Cholesky factorization

```
Do K = 1, N
    A(K, K) = SQRT(A(K, K))
    Do I = K+1, N
        A(I, K) = A(I, K) / A(K, K)
    Do I = K+1, N
        Do J = K+1, I
            A(I, J) = A(I, J) - A(I, K) * A(J, K)
```

# Key Insights and Design Details

- Loop distribution

  Example: Cholesky factorization

```
Do K = 1, N
    A(K, K) = SQRT(A(K, K))
    Do I = K+1, N
        A(I, K) = A(I, K) / A(K, K)
    Do J = K, N
        Do I = J+1, N
            A(I, J+1) = A(I, J+1) - A(I, K) * A(J+1, K)
```

# Key Insights and Design Details

- Compound loop transformations
  - Loop permutation
  - Loop reversal
  - Loop fusion
  - Loop distribution

# Key Insights and Design Details

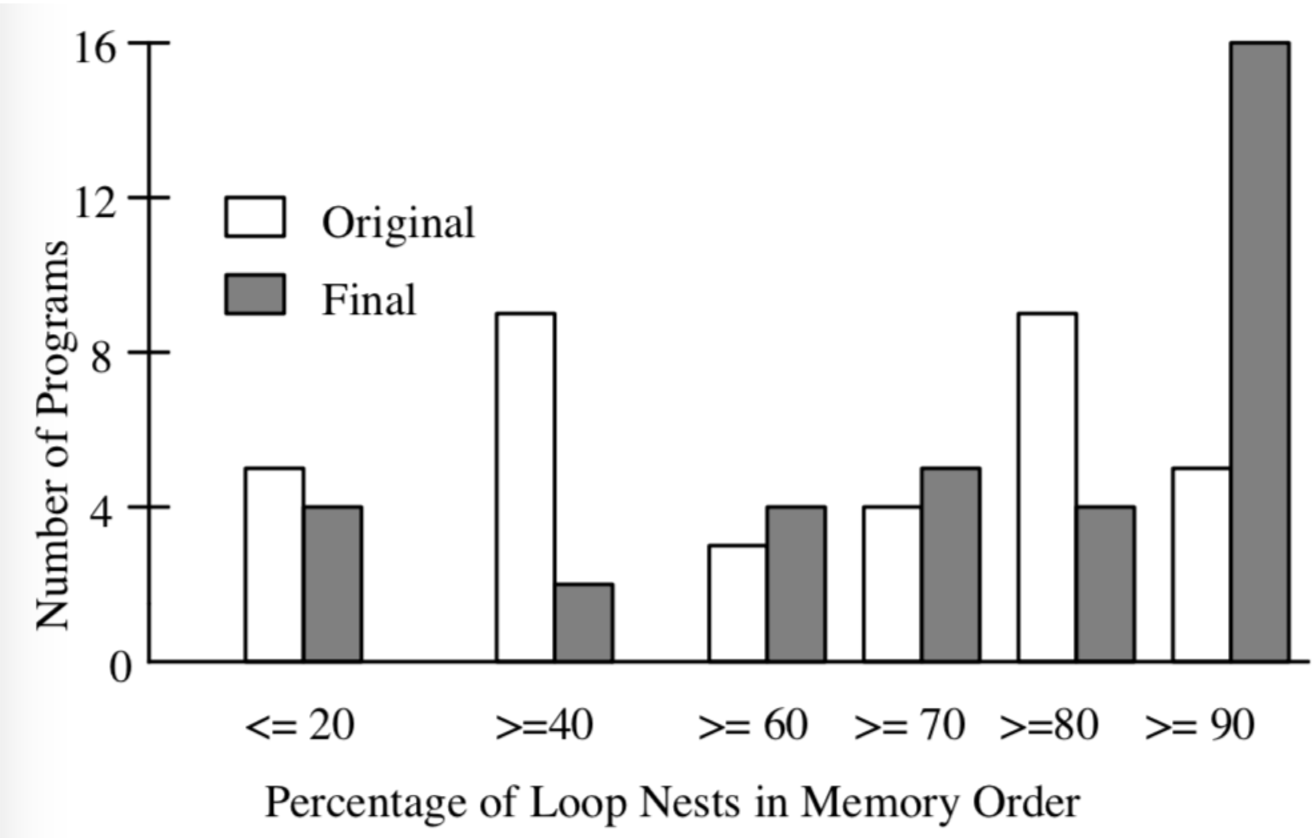
- Compound loop transformations
  - Do loop permutation for each loop nest
  - Try loop fusion and do loop permutation
  - Try loop distribution if helping permutation
  - Fuse all loop nests

# Key Results

- Analyze results in:
    - Number of programs achieving memory order
    - Speed
    - Cache hit rate

# Key Results

- Number of programs achieving memory order



# Key Results

- Speed

| Program | Original | Transformed | Speedup |
|---|---|---|---|
| *Perfect Benchmarks* | | | |
| arc2d | 410.13 | 190.69 | 2.15 |
| dyfesm | 25.42 | 25.37 | 1.00 |
| flo52 | 62.06 | 61.62 | 1.01 |
| *SPEC Benchmarks* | | | |
| dnasa7 (btrix) | 36.18 | 30.27 | 1.20 |
| dnasa7 (emit) | 16.46 | 16.39 | 1.00 |
| dnasa7 (gmtry) | 155.30 | 17.89 | 8.68 |
| dnasa7 (vpenta) | 149.68 | 115.62 | 1.29 |
| *NAS Benchmarks* | | | |
| applu | 146.61 | 149.49 | 0.98 |
| appsp | 361.43 | 337.84 | 1.07 |
| *Misc Programs* | | | |
| simple | 963.20 | 850.18 | 1.13 |
| linpackd | 159.04 | 157.48 | 1.01 |
| wave | 445.94 | 414.60 | 1.08 |

# Key Results

- Cache hit rate

| Program | Optimized Procedures Cache 1 Orig | Optimized Procedures Cache 1 Final | Optimized Procedures Cache 2 Orig | Optimized Procedures Cache 2 Final | Whole Program Cache 1 Orig | Whole Program Cache 1 Final | Whole Program Cache 2 Orig | Whole Program Cache 2 Final |
|---|---|---|---|---|---|---|---|---|
| *Perfect Benchmarks* | | | | | | | | |
| adm | 100 | 100 | **97.7** | **97.8** | 99.95 | 99.95 | **98.48** | **98.58** |
| arc2d | **89.0** | **98.5** | **68.3** | **91.9** | **95.30** | **98.66** | **88.58** | **93.61** |
| bdna | 100 | 100 | 100 | 100 | 99.45 | 99.45 | 97.32 | 97.32 |
| dyfesm | 100 | 100 | 100 | 100 | 99.98 | 99.97 | 97.02 | 96.95 |
| flo52 | 99.6 | 99.6 | **96.7** | **96.3** | 98.77 | 98.77 | 93.84 | 93.80 |
| mdg | 100 | 100 | 87.4 | 87.4 | —— | —— | —— | —— |
| mg3d | **98.8** | **99.7** | **95.3** | **98.7** | —— | —— | —— | —— |
| ocean | 100 | 100 | **93.0** | **92.8** | 99.36 | 99.36 | 93.71 | 93.72 |
| qcd | 100 | 100 | 100 | 100 | 99.83 | 99.83 | 98.85 | 98.79 |
| spec77 | 100 | 100 | 100 | 100 | 99.28 | 99.28 | 93.79 | 93.78 |
| track | 100 | 100 | 100 | 100 | 99.81 | 99.81 | 97.49 | 97.54 |
| trfd | 99.9 | 99.9 | 93.7 | 93.7 | 99.92 | 99.92 | 96.43 | 96.40 |

# Key Results

- Analyze results in:
  - Number of programs achieving memory order
  - Speed
  - Cache hit rate
- Prove effect of loop fusion and distribution

# Strengths

- Introduce loop fusion & distribution & reversal
  - Improve performance of loop permutation
  - Permit imperfect loop nests
- Proper reference group formulation

# Weaknesses

- Limitation in uniprocessor system
- Limitation in rectangular and triangular nests

# Open Questions

- Loop tiling
- Loop fusion’s defect

# Discussion

Thank you!