
Uncovering Hidden Loop Level Parallelism in Sequential Applications

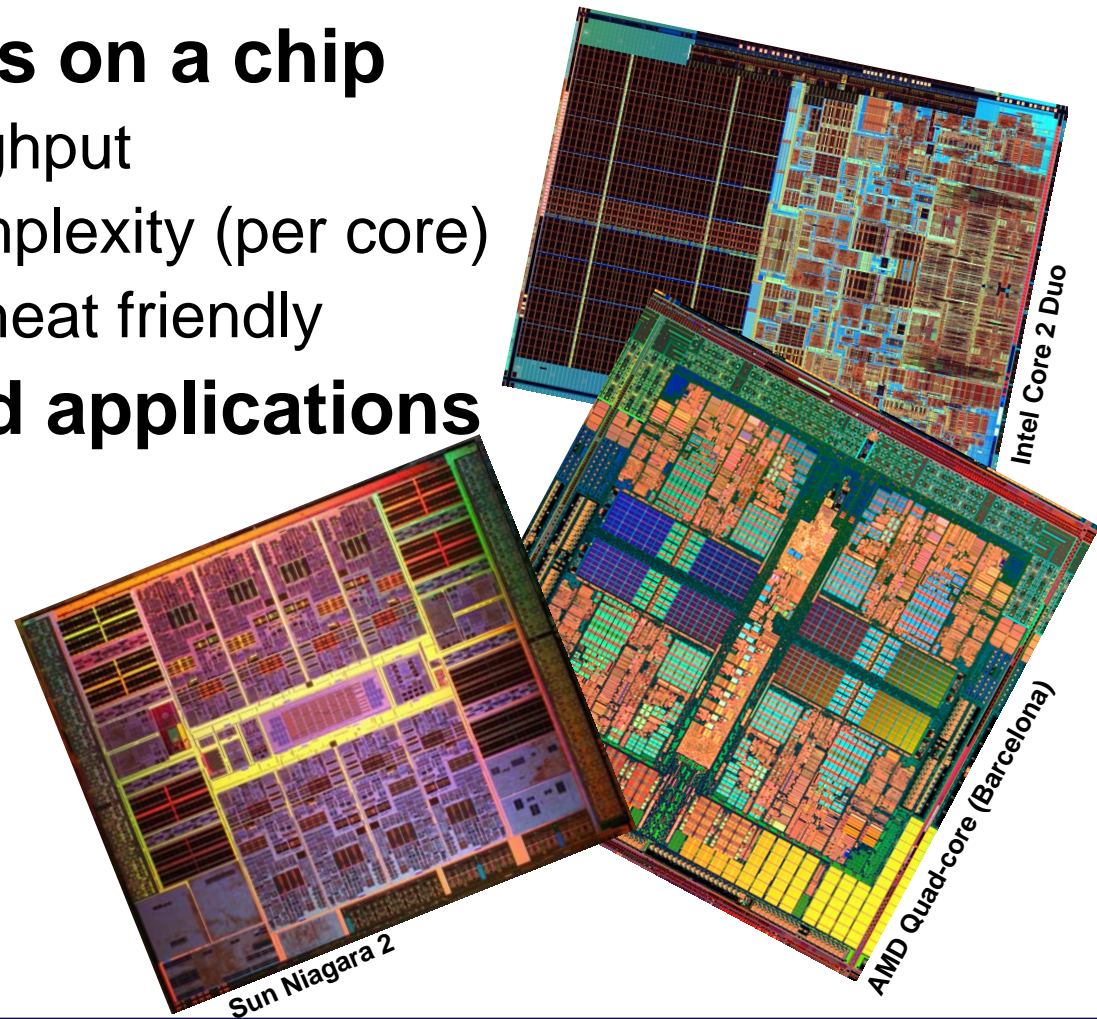
Hongtao Zhong, Mojtaba Mehrara, Steve Lieberman,
Scott Mahlke

Advanced Computer Architecture Lab.
University of Michigan

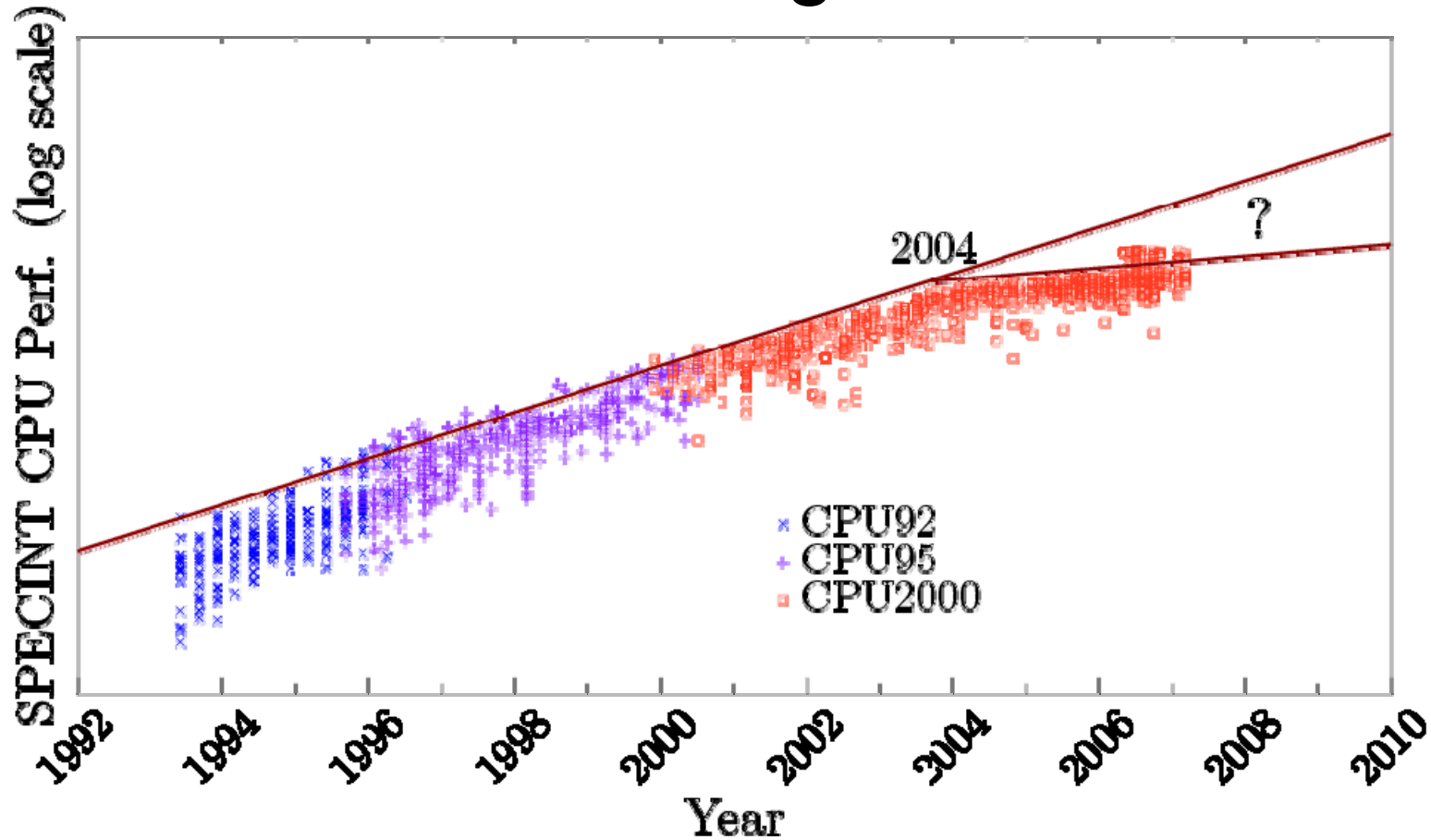


CMP Architectures

- **Multiple cores on a chip**
 - Higher throughput
 - Reduced complexity (per core)
 - More power/heat friendly
- **Multithreaded applications**



How About Single Thread?

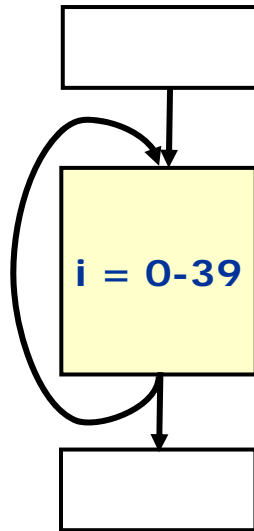


[Source : Bridges et al, MICRO `07]



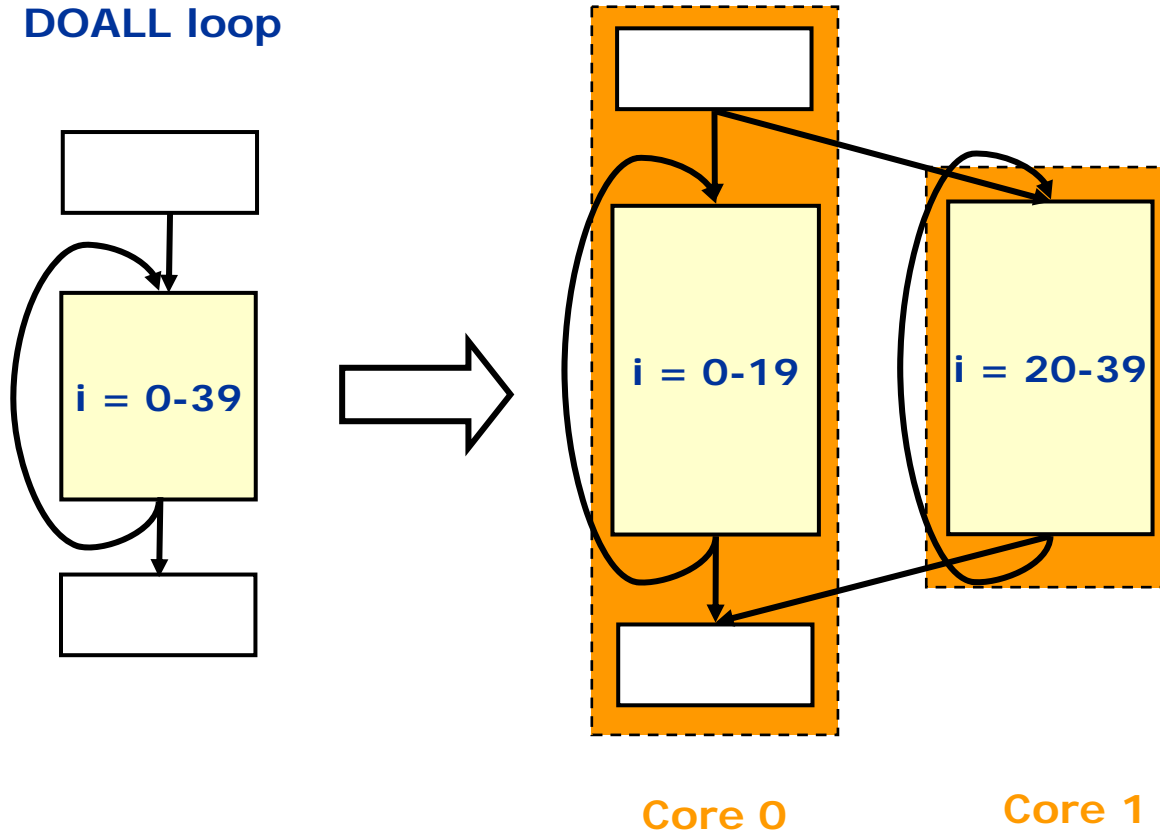
Loop Level Parallelization

DOALL loop



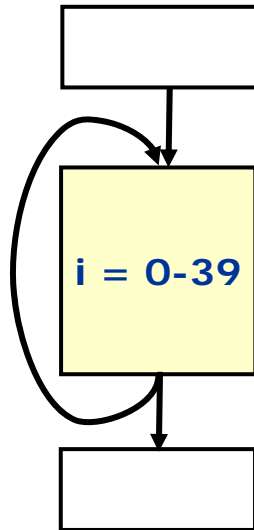
Loop Level Parallelization

DOALL loop



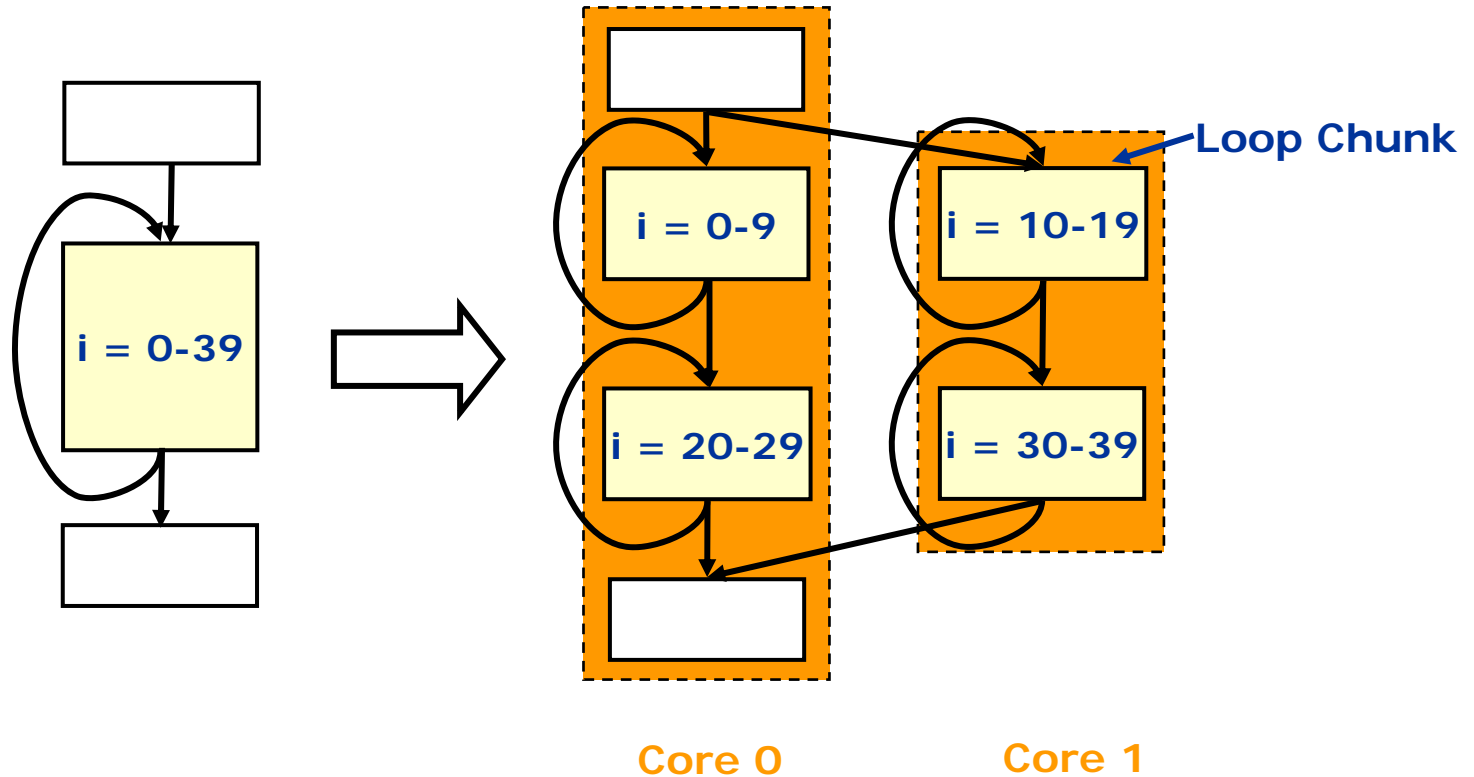
Loop Level Parallelization

Speculative DOALL loop



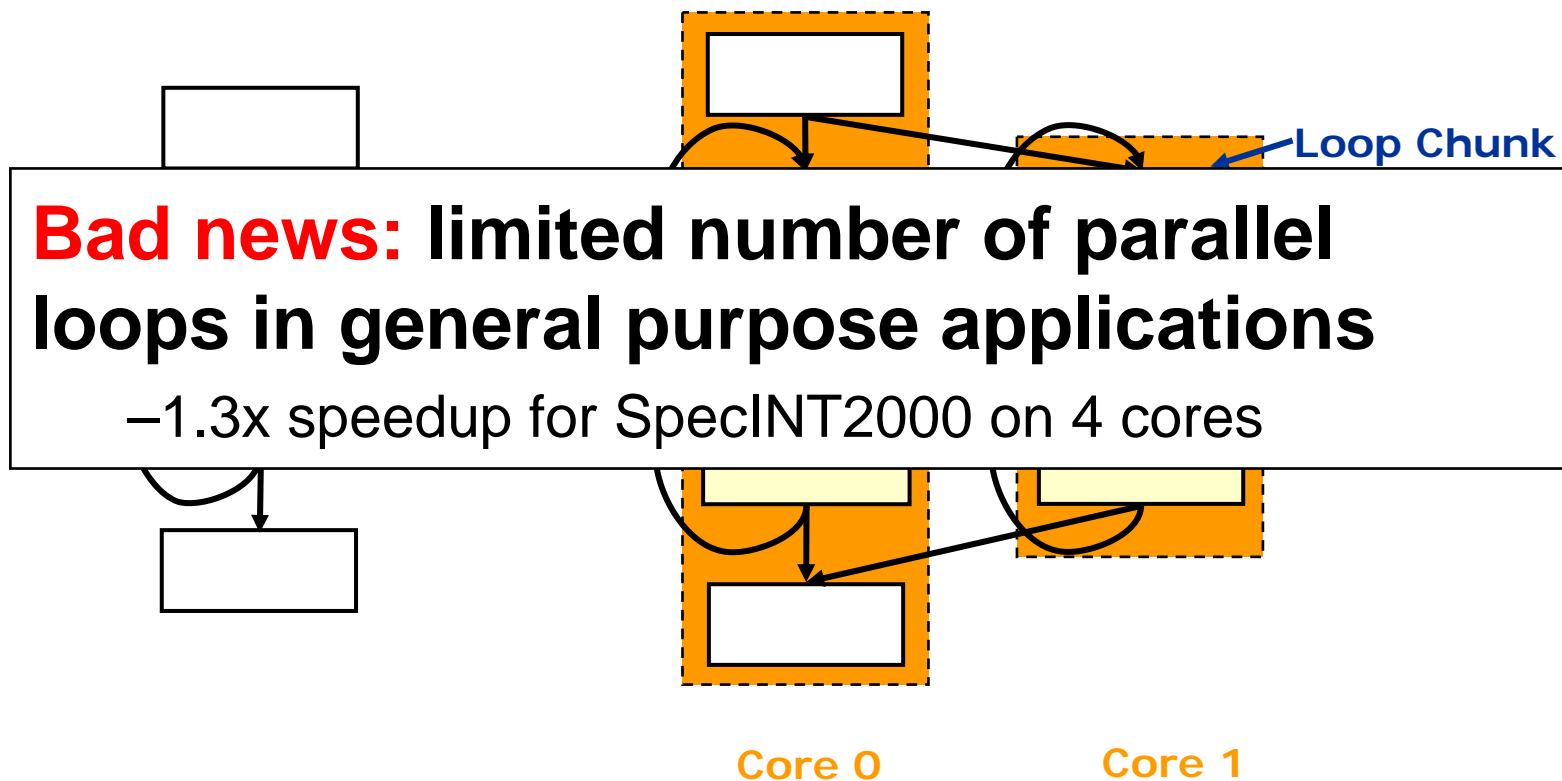
Loop Level Parallelization

Speculative DOALL loop



Loop Level Parallelization

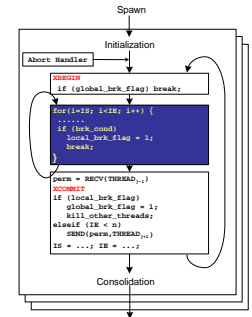
Speculative DOALL loop



Contributions

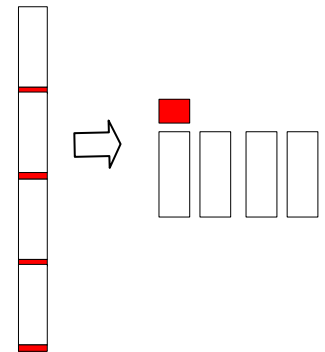
- **Code generation framework**

- Speculative parallelization of uncounted loops

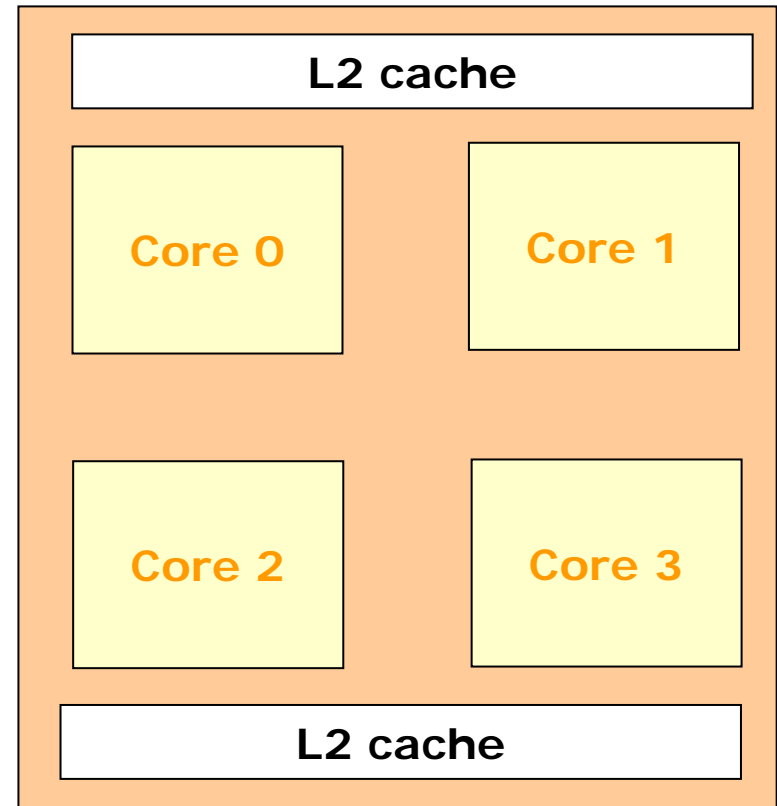


- **Compiler transformations**

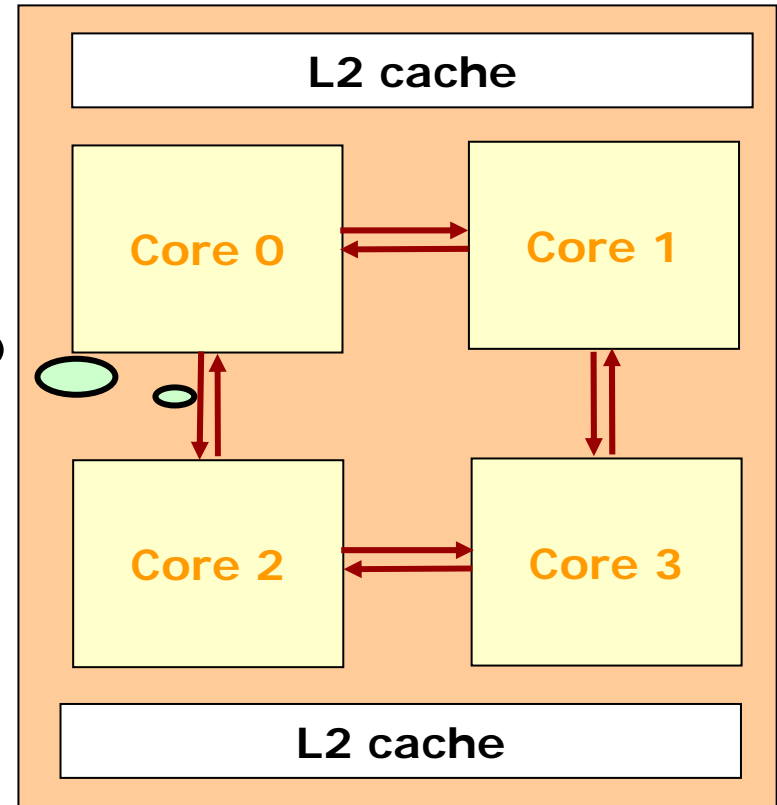
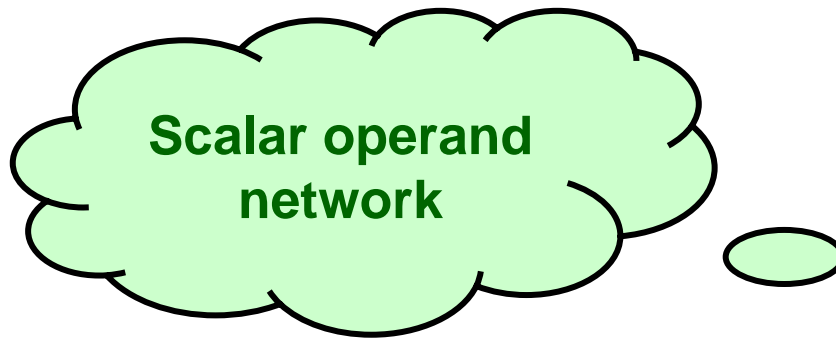
- Speculative loop fission
- Isolation of infrequent dependences
- Speculative prematerialization



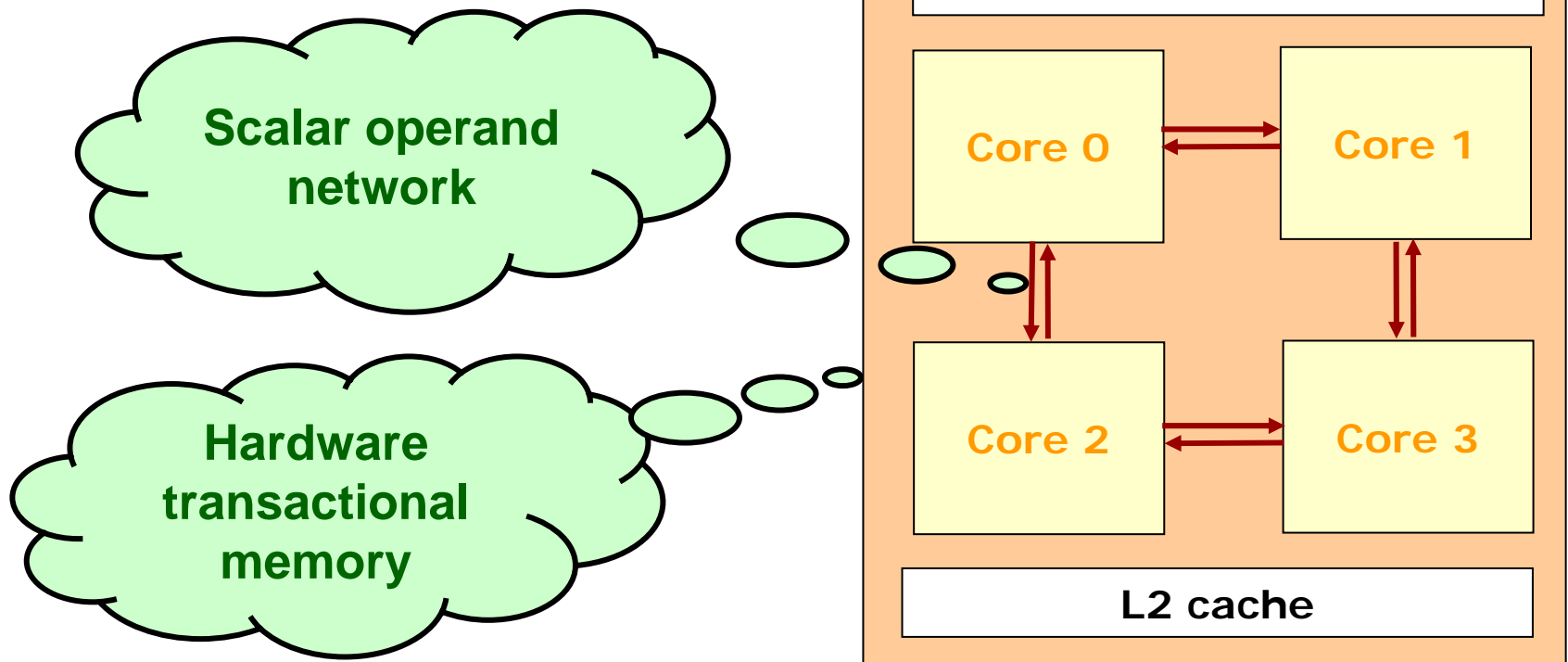
Target Architecture



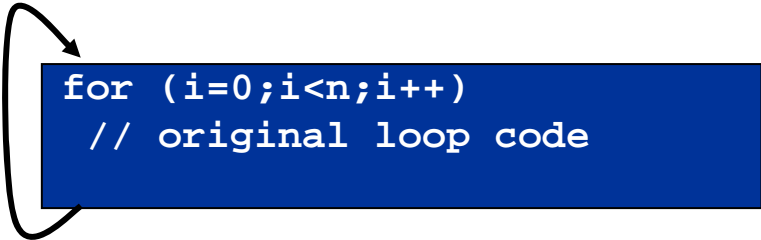
Target Architecture



Target Architecture

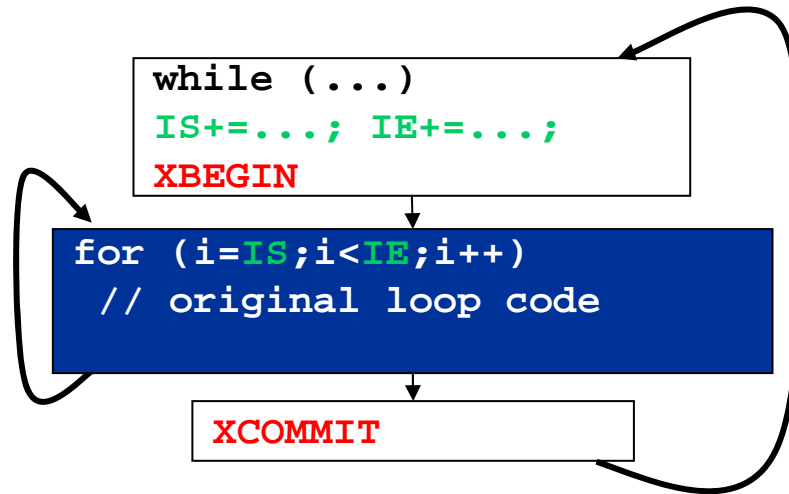


Code Generation Framework

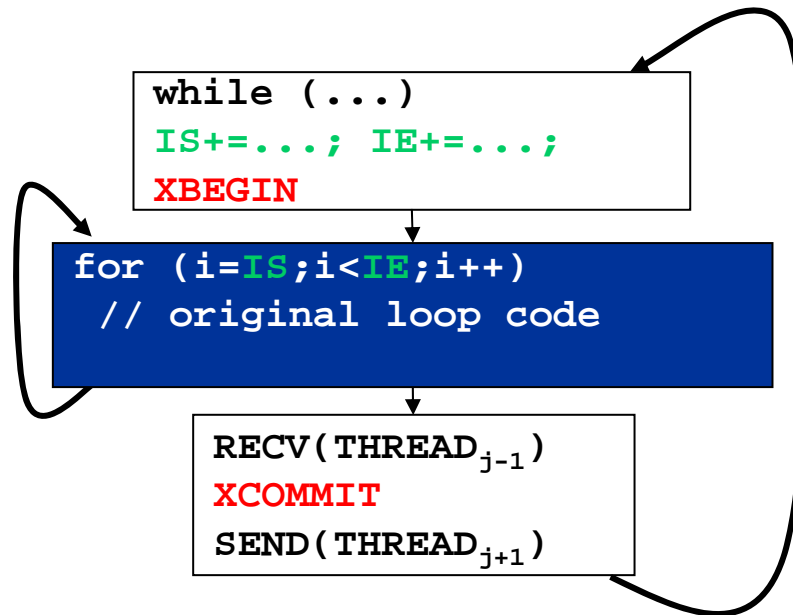


```
for (i=0;i<n;i++)  
  // original loop code
```

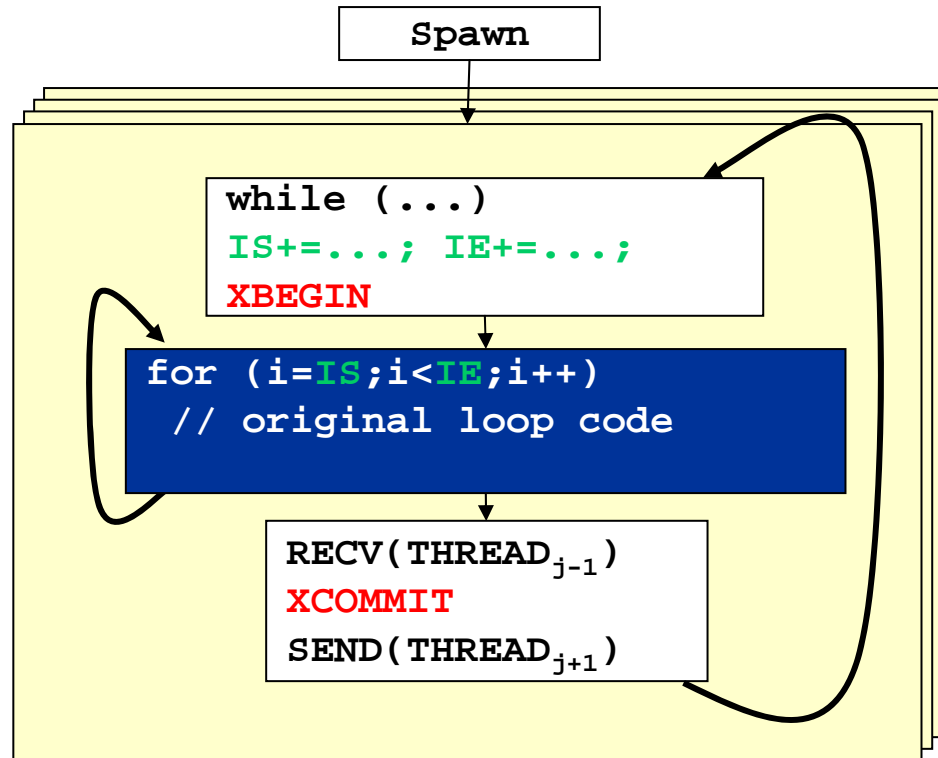
Code Generation Framework



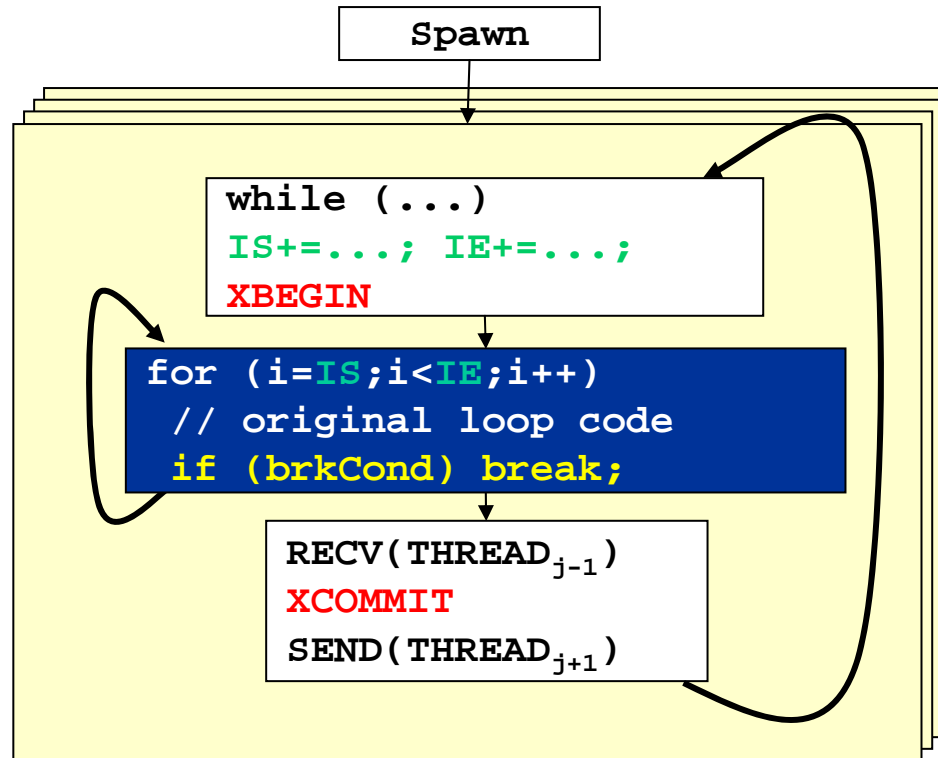
Code Generation Framework



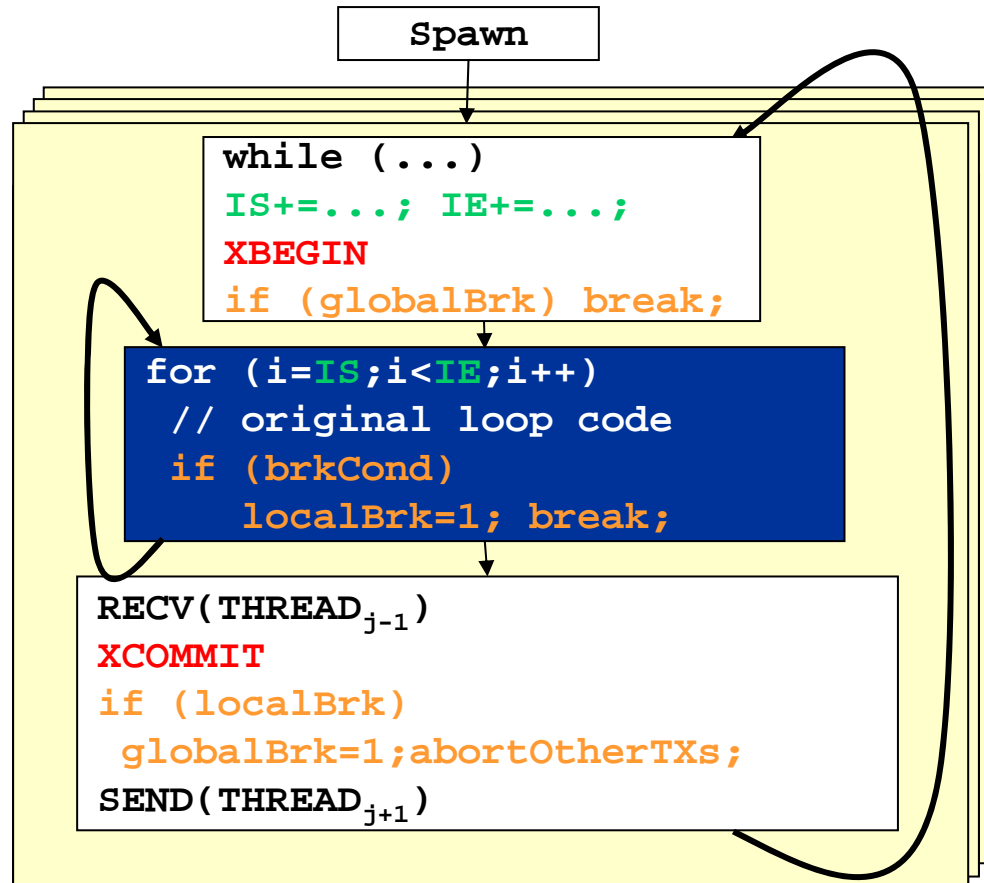
Code Generation Framework



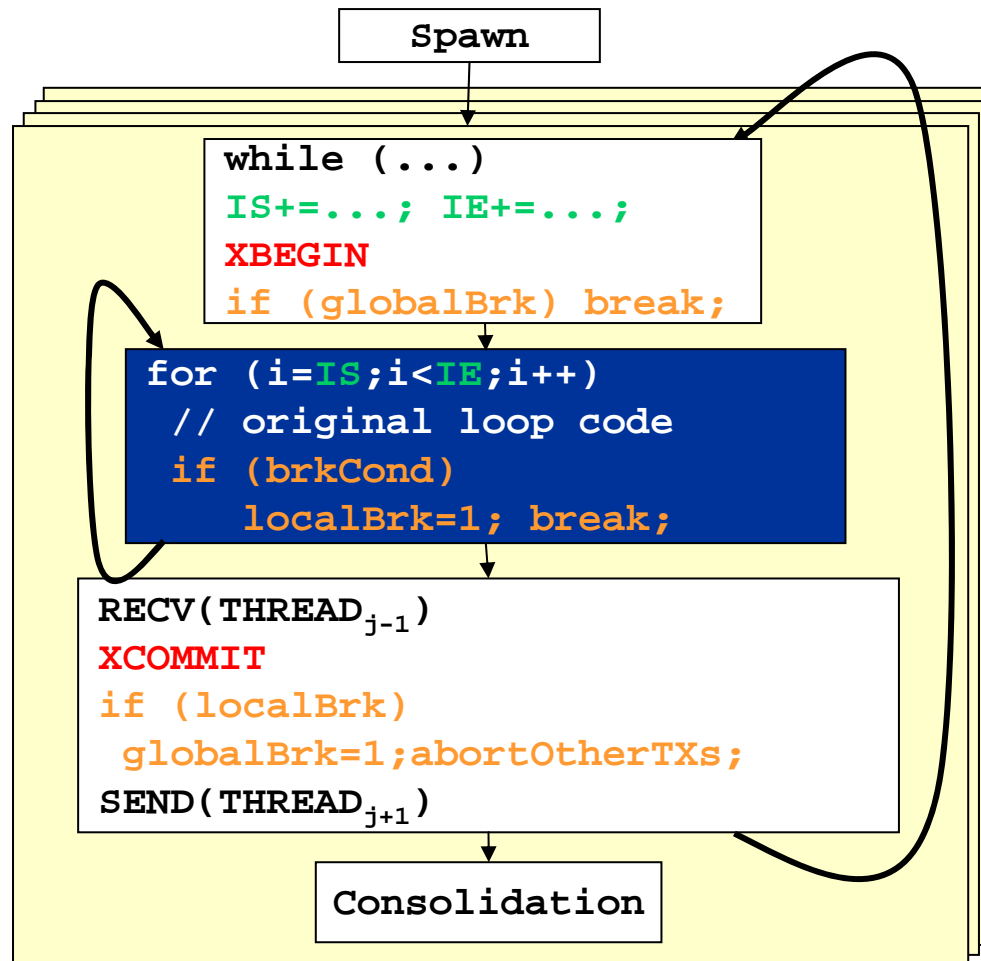
Code Generation Framework



Code Generation Framework

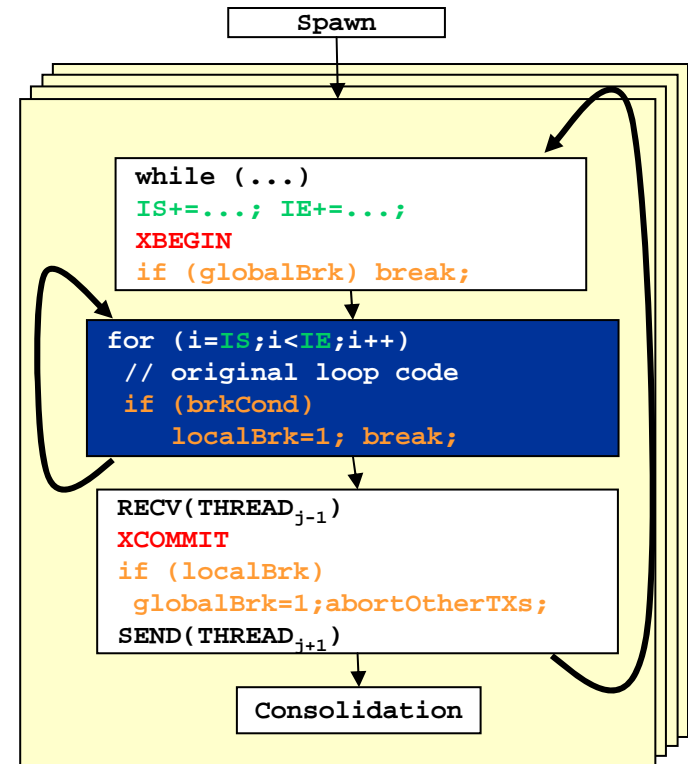


Code Generation Framework

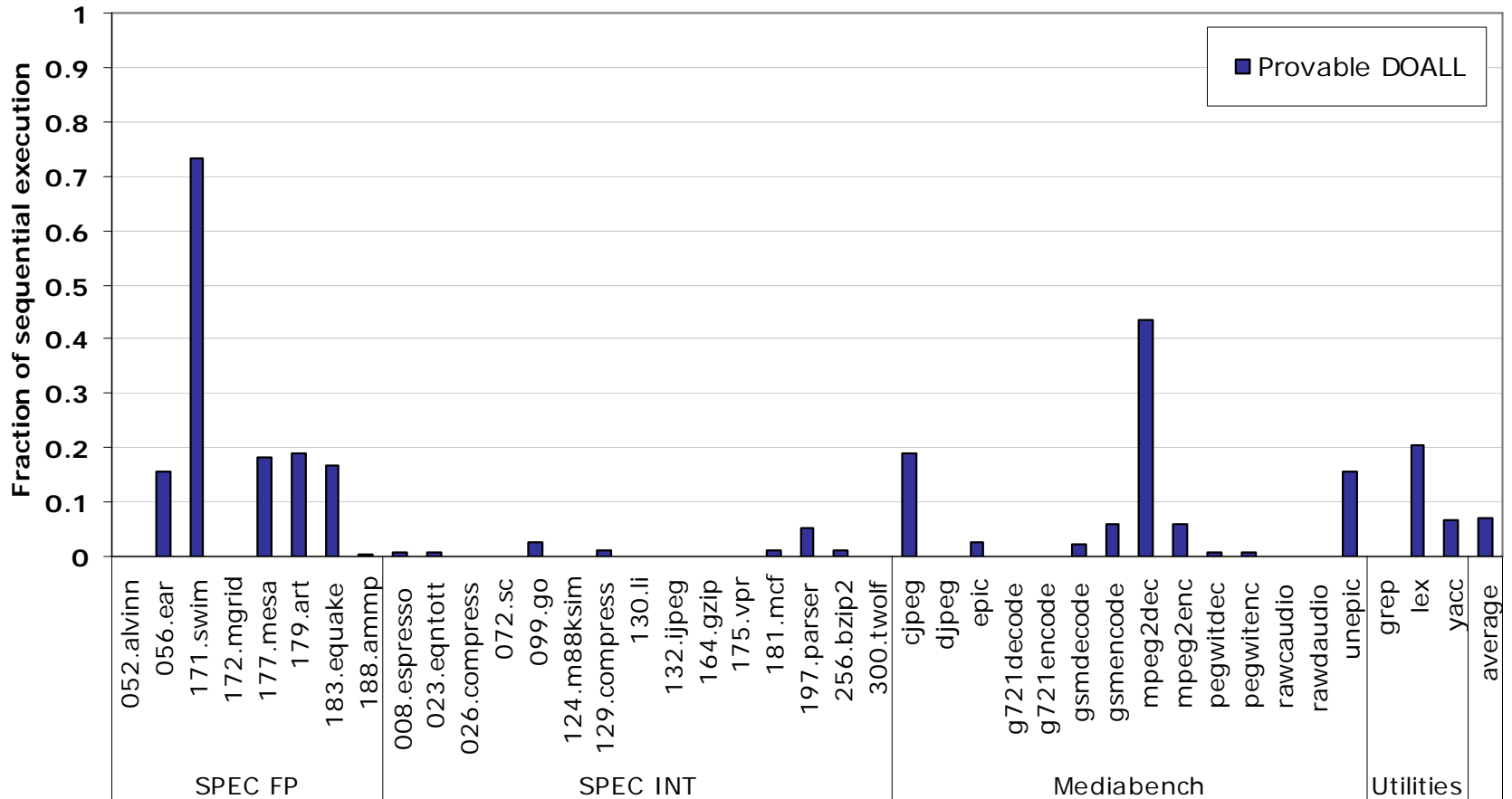


Code Generation Framework

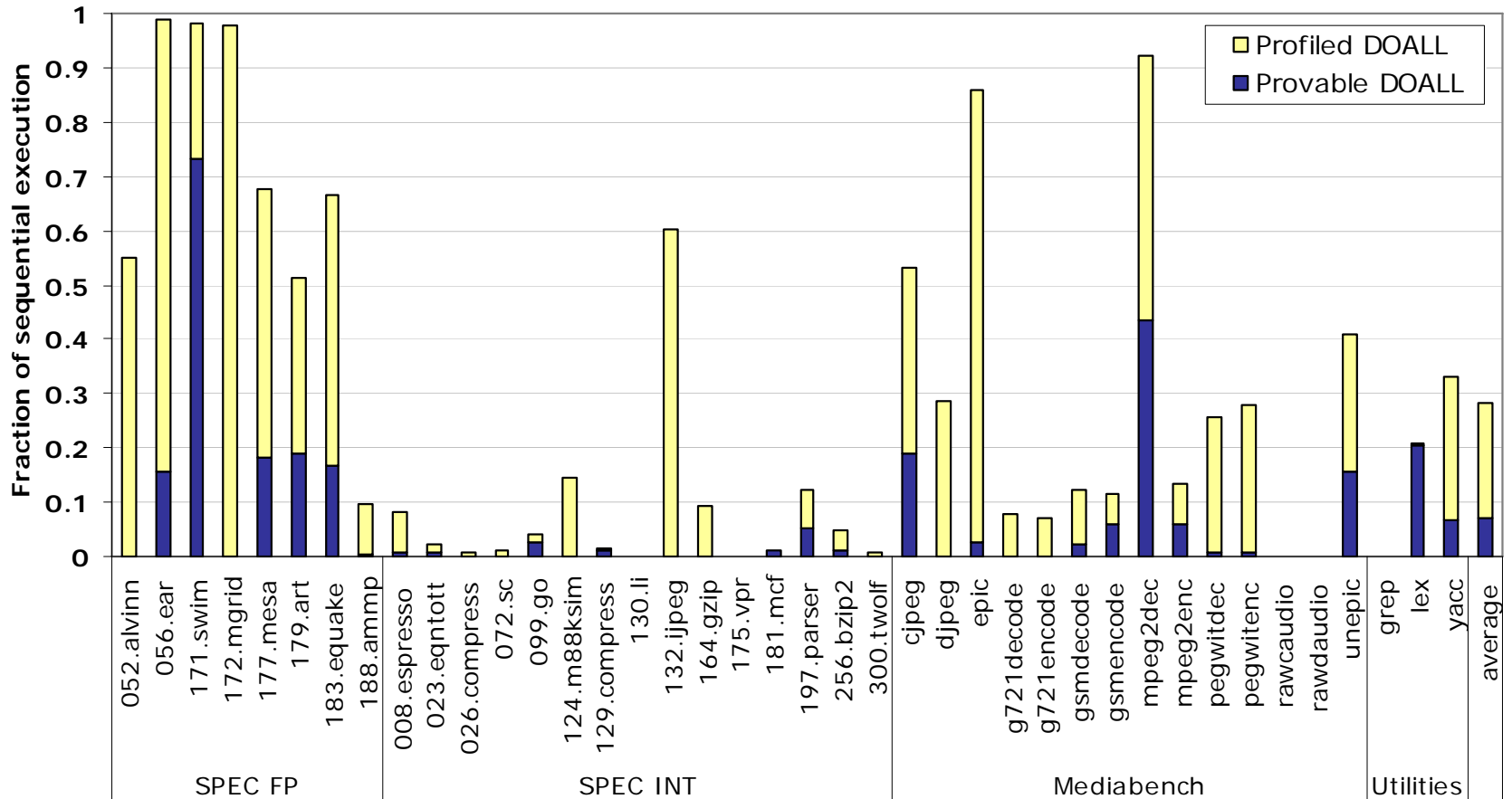
- Supports **counted** and **uncounted** loops
 - Software managed control speculation
- Iteration chunking
- Enforce transaction ordering
- Handles livein, liveout & accumulator registers



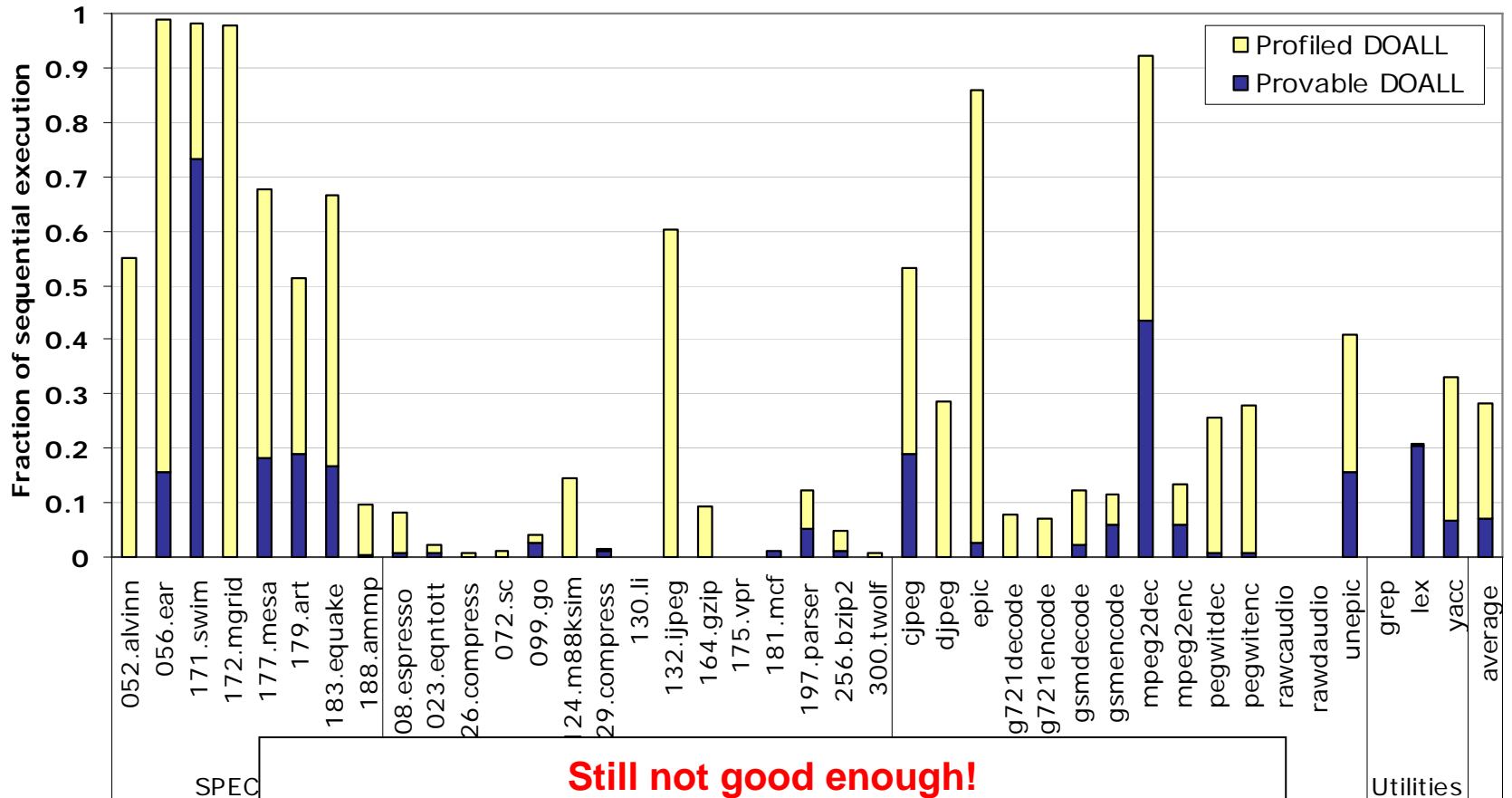
DOALL Coverage – Provable and Profiled



DOALL Coverage – Provable and Profiled



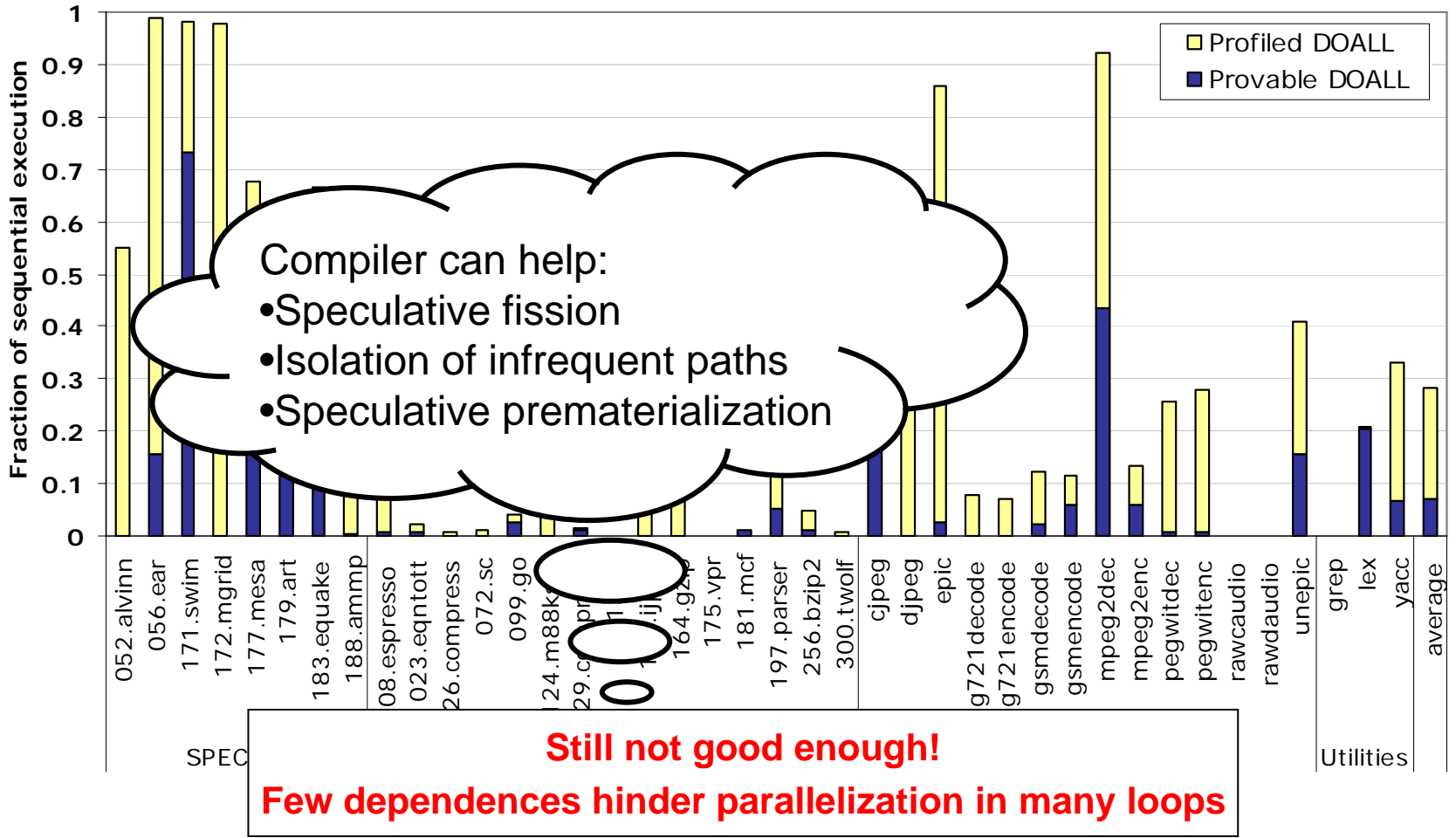
DOALL Coverage – Provable and Profiled



Still not good enough!
Few dependences hinder parallelization in many loops




DOALL Coverage – Provable and Profiled



Speculative Loop Fission

```
1: while (node) {  
2:   work(node);  
3:   node = node->next;  
}
```

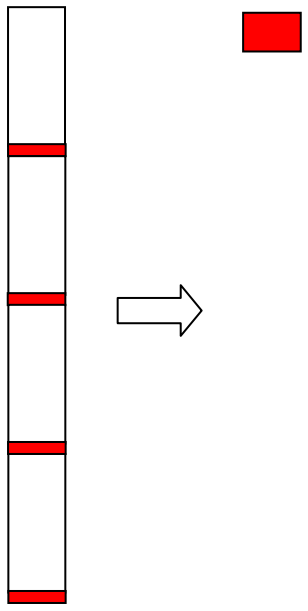


Speculative Loop Fission

```
1: while (node) {  
2:   work(node);  
3:   node = node->next;  
}
```



```
1: while (node) {  
4:   node_array[count++] = node;  
3:   node = node->next;  
}
```

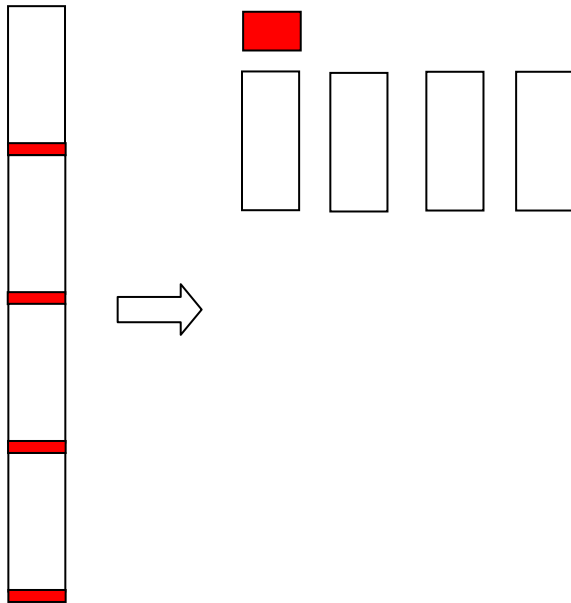


Speculative Loop Fission

```
1: while (node) {  
2:   work(node);  
3:   node = node->next;  
}
```



```
1: while (node) {  
4:   node_array[count++] = node;  
3:   node = node->next;  
}
```



```
XBEGIN  
5: node = node_array[IS];  
   i = 0;  
1':while (node && i++ < CS) {  
2:   work(node);  
3':   node = node->next;  
   }  
   RECV(THREADj-1)  
   XCOMMIT  
   SEND(THREADj+1)  
}
```



Speculative Loop Fission

```

1: while (node) {
2:   work(node);
3:   node = node->next;
}

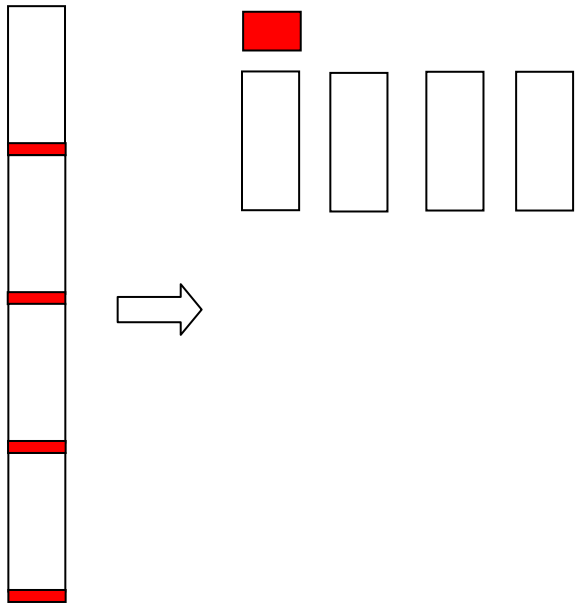
```



```

1: while (node) {
4:   node_array[count++] = node;
3:   node = node->next;
}

```



```

XBEGIN
5: node = node_array[IS];
   i = 0;
1':while (node && i++ < CS) {
2:   work(node);
3':   node = node->next;
}
RECV(THREADj-1)
XCOMMIT
if (node != node_array[IS+CS]){
   update_node_array;
   kill_other_threads();}
SEND(THREADj+1)
}

```




Speculative Loop Fission

```

1: while (node) {
2:   work(node);
3:   node = node->next;
}

```




```

1: while (node) {
4:   node_array[count++] = node;
3:   node = node->next;
}

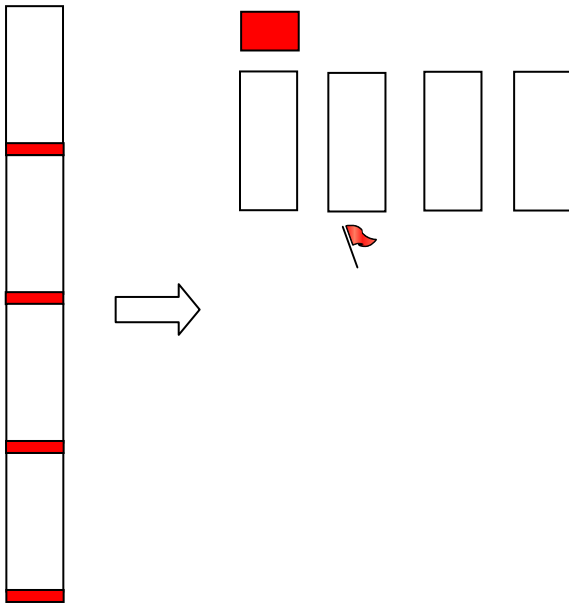
```



```

XBEGIN
5: node = node_array[IS];
   i = 0;
1':while (node && i++ < CS) {
2:   work(node);
3':   node = node->next;
}
RECV(THREADj-1)
XCOMMIT
if (node != node_array[IS+CS]){
  update_node_array;
  kill_other_threads();}
SEND(THREADj+1)
}

```



Speculative Loop Fission

```

1: while (node) {
2:   work(node);
3:   node = node->next;
}

```



```

1: while (node) {
4:   node_array[count++] = node;
3:   node = node->next;
}

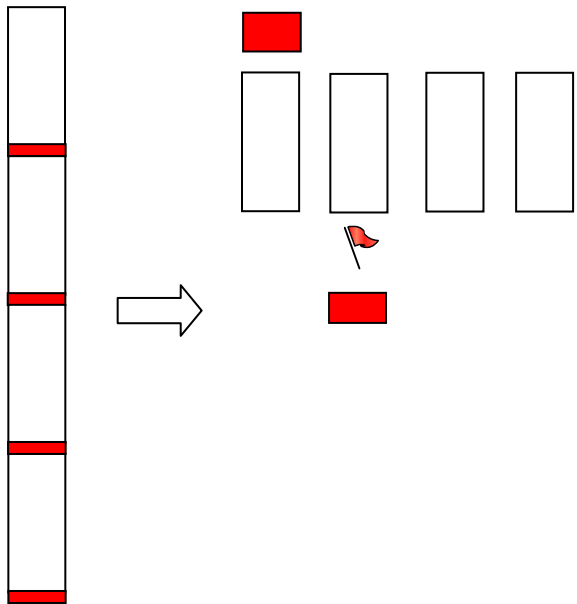
```



```

XBEGIN
5: node = node_array[IS];
   i = 0;
1':while (node && i++ < CS) {
2:   work(node);
3':   node = node->next;
}
RECV(THREADj-1)
XCOMMIT
if (node != node_array[IS+CS]){
  update_node_array;
  kill_other_threads();}
SEND(THREADj+1)
}

```



Speculative Loop Fission

```

1: while (node) {
2:   work(node);
3:   node = node->next;
}
    
```



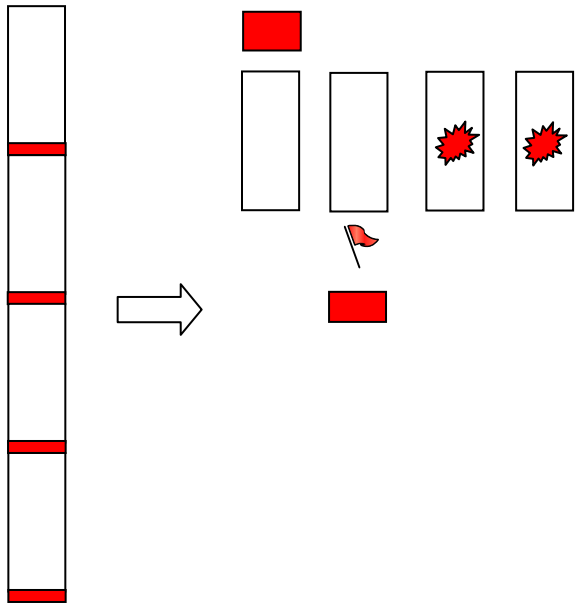
```

1: while (node) {
4:   node_array[count++] = node;
3:   node = node->next;
}
    
```



```

XBEGIN
5: node = node_array[IS];
   i = 0;
1':while (node && i++ < CS) {
2:   work(node);
3':   node = node->next;
   }
   RECV(THREADj-1)
   XCOMMIT
   if (node != node_array[IS+CS]){
       update_node_array;
       kill_other_threads();}
   SEND(THREADj+1)
}
    
```




Speculative Loop Fission

```

1: while (node) {
2:   work(node);
3:   node = node->next;
}

```




```

1: while (node) {
4:   node_array[count++] = node;
3:   node = node->next;
}

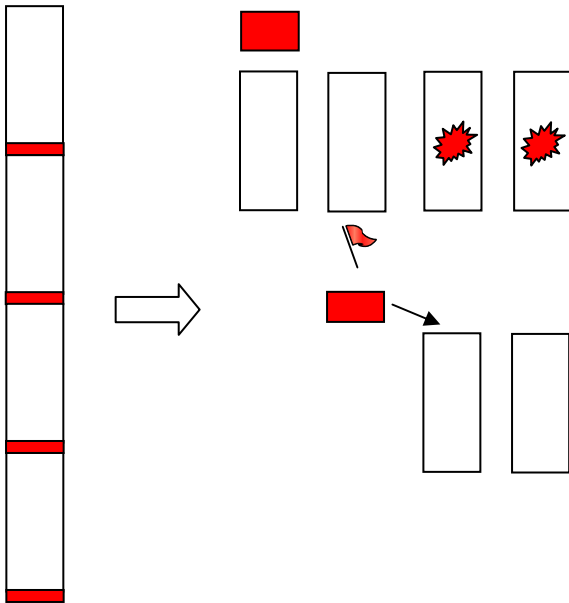
```



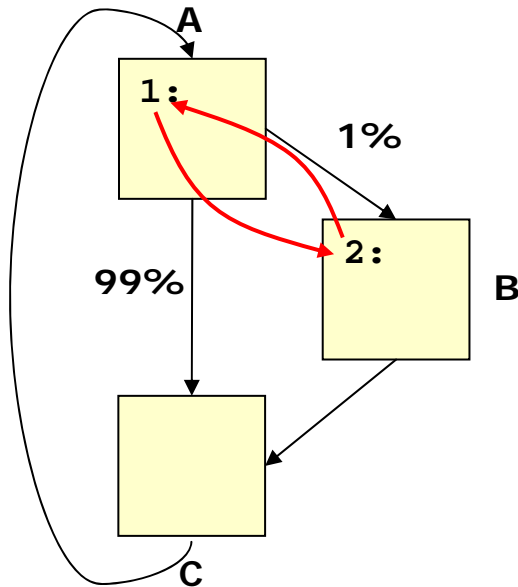
```

XBEGIN
5: node = node_array[IS];
   i = 0;
1':while (node && i++ < CS) {
2:   work(node);
3':   node = node->next;
}
RECV(THREADj-1)
XCOMMIT
if (node != node_array[IS+CS]){
  update_node_array;
  kill_other_threads();}
SEND(THREADj+1)
}

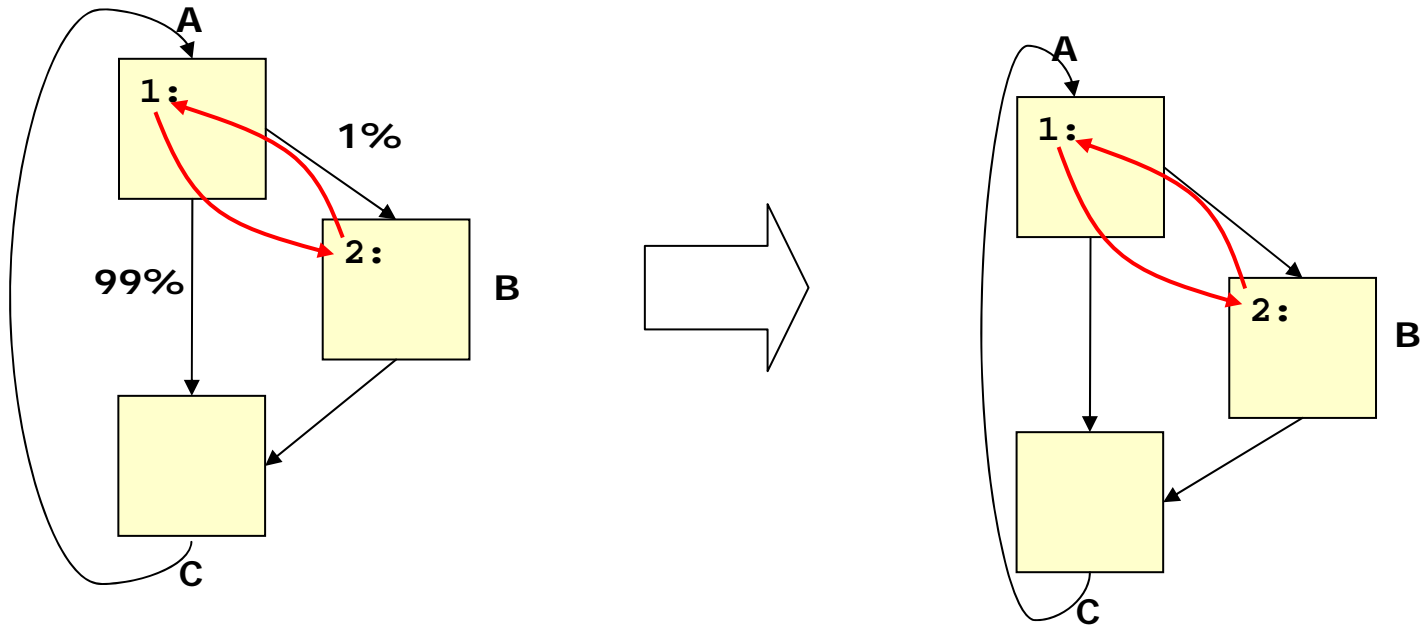
```



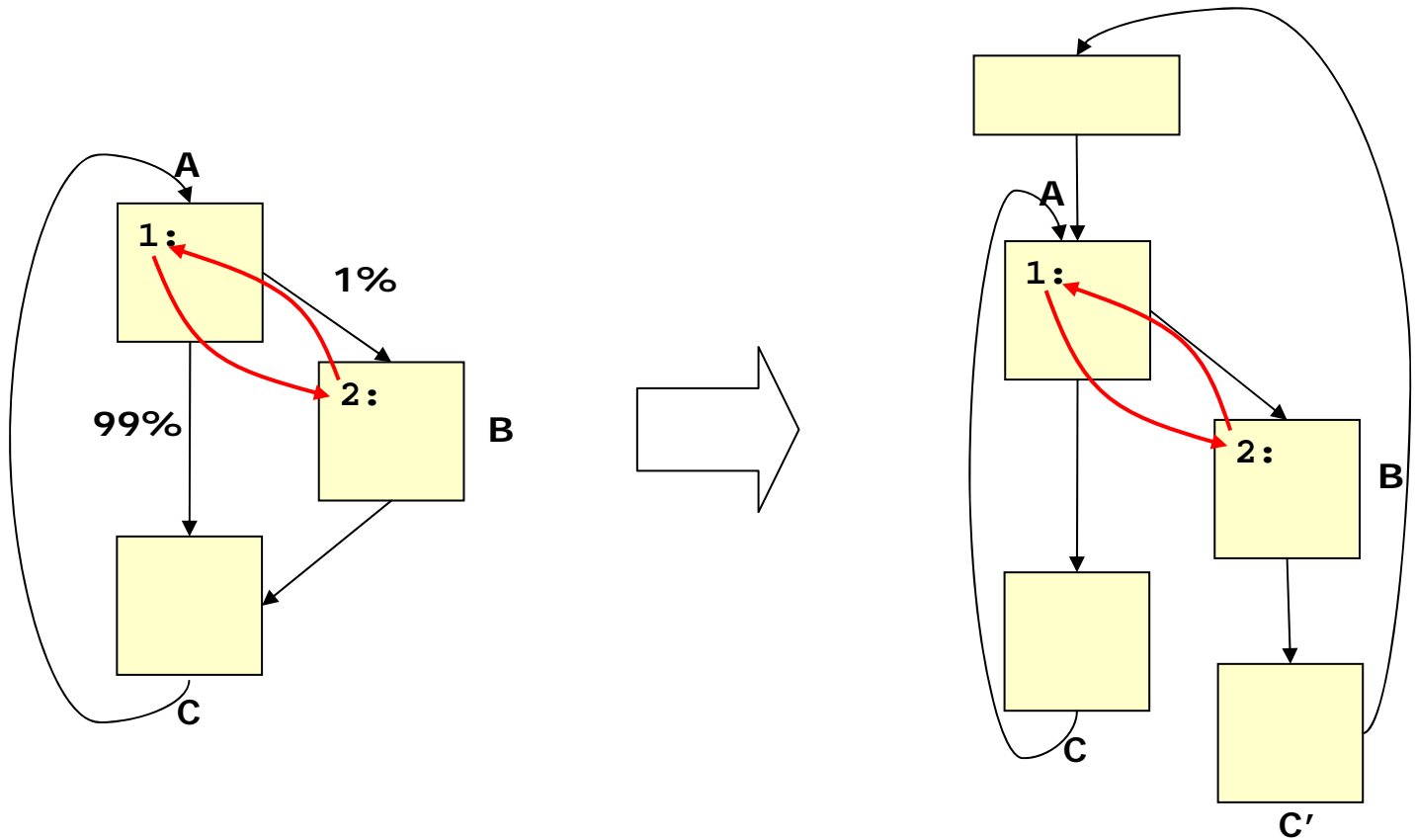
Infrequent Dependence Isolation



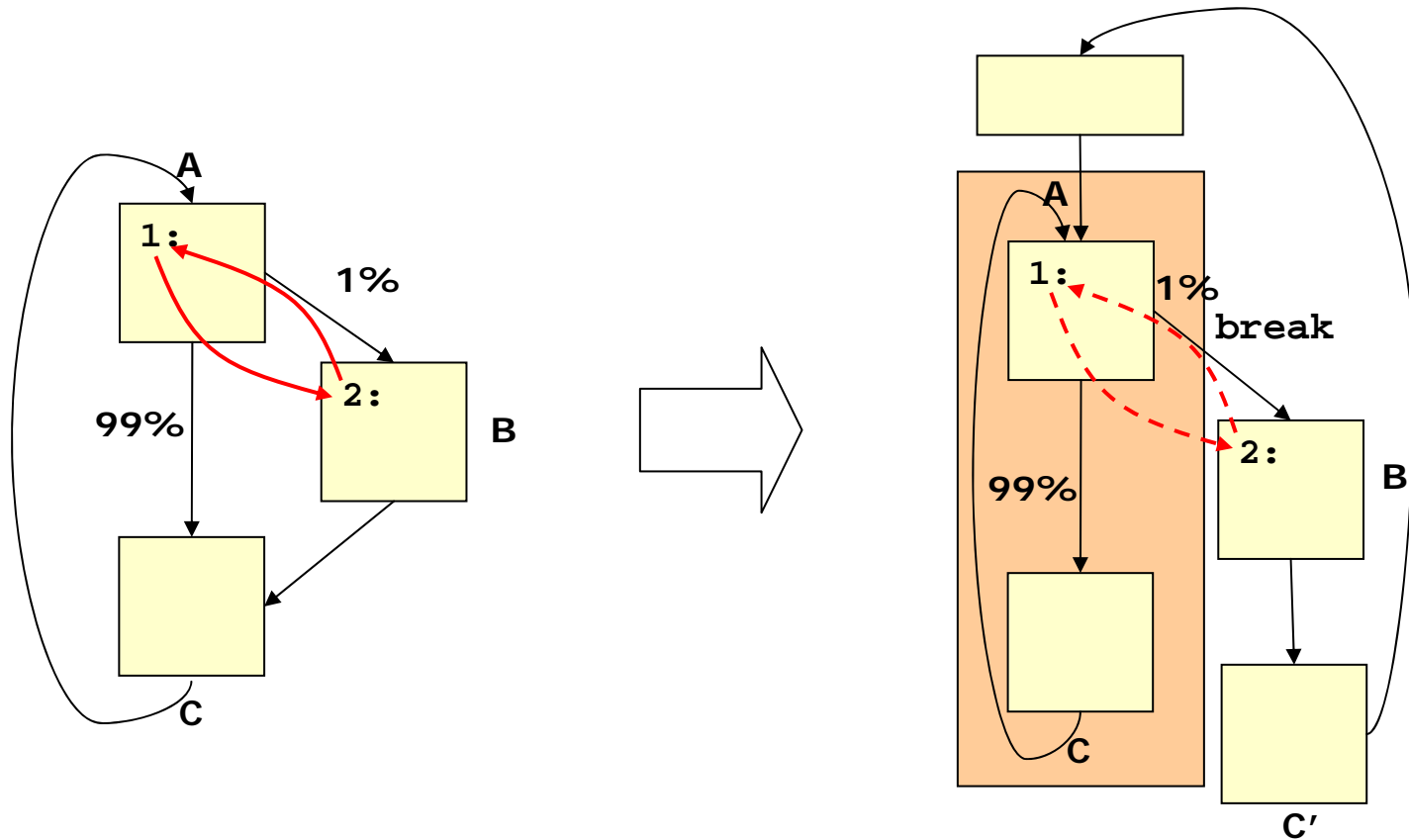
Infrequent Dependence Isolation



Infrequent Dependence Isolation



Infrequent Dependence Isolation



Infrequent Dependence Isolation

```
for( j=0; j<=nstate; ++j ){
  if( tystate[j] == 0 ) continue;
  if( tystate[j] == best ) continue;
  count = 0;
  cbest = tystate[j];
  for (k=j; k<=nstate; ++k)
    if (tystate[k]==cbest) ++count;
  if ( count > times) {
    best = cbest;
    times = count;
  }
}
```

Sample loop from yacc benchmark



Infrequent Dependence Isolation

```
for( j=0; j<=nstate; ++j ){
  if( tystate[j] == 0 ) continue;
  if( tystate[j] == best ) continue;
  count = 0;
  cbest = tystate[j];
  for (k=j; k<=nstate; ++k)
    if (tystate[k]==cbest) ++count;
  if ( count > times) {
    best = cbest;
    times = count;
  }
}
```

Sample loop from yacc benchmark



Infrequent Dependence Isolation

```
for( j=0; j<=nstate; ++j ){
  if( tystate[j] == 0 ) continue;
  if( tystate[j] == best ) continue;
  count = 0;
  cbest = tystate[j];
  for (k=j; k<=nstate; ++k)
    if (tystate[k]==cbest) ++count;
  if (count > times) {
    best = cbest;
    times = count;
  }
}
```

1 %

Sample loop from yacc benchmark



Infrequent Dependence Isolation

```
for( j=0; j<=nstate; ++j ){
  if( tystate[j] == 0 ) continue;
  if( tystate[j] == best ) continue;
  count = 0;
  cbest = tystate[j];
  for (k=j; k<=nstate; ++k)
    if (tystate[k]==cbest) ++count;
  if ( count > times ) {
    best = cbest;
    times = count;
  }
}
```

1 %

Sample loop from yacc benchmark

```
j=0;
while (j<=nstate){
  for( ; j<=nstate; ++j ){
    if( tystate[j] == 0 ) continue;
    if( tystate[j] == best ) continue;
    count = 0;
    cbest = tystate[j];
    for (k=j; k<=nstate; ++k)
      if (tystate[k]==cbest) ++count;
    if ( count > times )
      break;
  }
  if ( count > times ) {
    best = cbest;
    times = count; j++;
  }
}
```

1 %



Infrequent Dependence Isolation

```
for( j=0; j<=nstate; ++j ){
  if( tystate[j] == 0 ) continue;
  if( tystate[j] == best ) continue;
  count = 0;
  cbest = tystate[j];
  for (k=j; k<=nstate; ++k)
    if (tystate[k]==cbest) ++count;
  if (count > times) {
    best = cbest;
    times = count;
  }
}
```

1 %

Sample loop from yacc benchmark

```
j=0;
while (j<=nstate){
  for( ; j<=nstate; ++j ){
    if( tystate[j] == 0 ) continue;
    if( tystate[j] == best ) continue;
    count = 0;
    cbest = tystate[j];
    for (k=j; k<=nstate; ++k)
      if (tystate[k]==cbest) ++count;
    if (count > times)
      break;
  }
  if (count > times) {
    best = cbest;
    times = count; j++;
  }
}
```

1 %



Infrequent Dependence Isolation

```
for( j=0; j<=nstate; ++j ){
  if( tystate[j] == 0 ) continue;
  if( tystate[j] == best ) continue;
  count = 0;
  cbest = tystate[j];
  for (k=j; k<=nstate; ++k)
    if (tystate[k]==cbest) ++count;
  if ( count > times ) {
    best = cbest;
    times = count;
  }
}
```

1 %

Sample loop from yacc benchmark

```
j=0;
while (j<=nstate){
  for( ; j<=nstate; ++j ){
    if( tystate[j] == 0 ) continue;
    if( tystate[j] == best ) continue;
    count = 0;
    cbest = tystate[j];
    for (k=j; k<=nstate; ++k)
      if (tystate[k]==cbest) ++count;
    if ( count > times )
      break;
  }
  if ( count > times ) {
    best = cbest;
    times = count; j++;
  }
}
```

1 %

1 %



Infrequent Dependence Isolation

```
for( j=0; j<=nstate; ++j ){  
  if( tystate[j] == 0 ) continue;  
  if( tystate[j] == best ) continue;  
  count = 0;  
  cbest = tystate[j];  
  for (k=j; k<=nstate; ++k)  
    if (tystate[k]==cbest) ++count;  
  if ( count > times ) {  
    best = cbest;  
    times = count;  
  }  
}
```

1 %

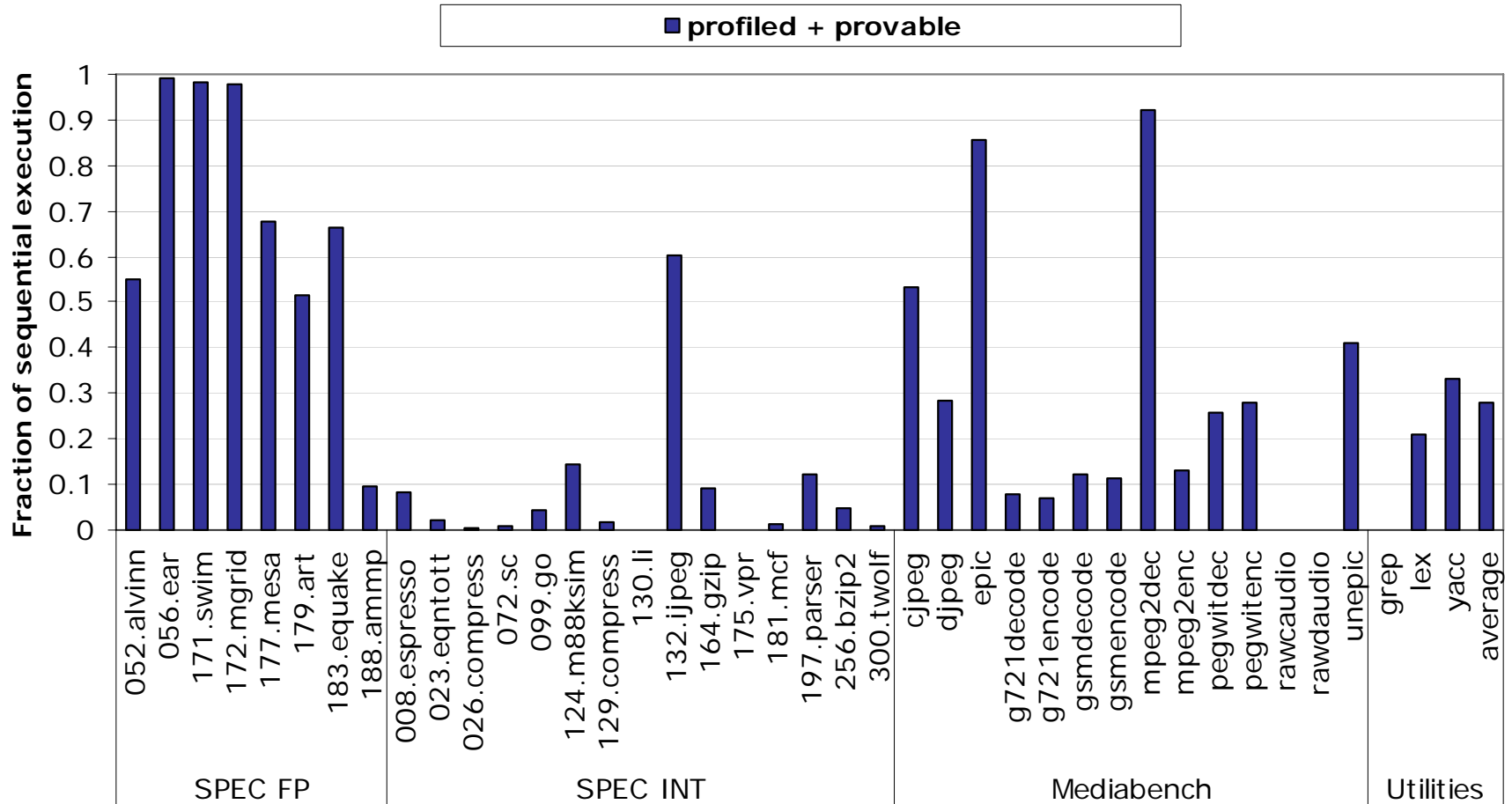
Sample loop from yacc benchmark

```
j=0;  
while (j<=nstate){  
  for( ; j<=nstate; ++j ){  
    if( tystate[j] == 0 ) continue;  
    if( tystate[j] == best ) continue;  
    count = 0;  
    cbest = tystate[j];  
    for (k=j; k<=nstate; ++k)  
      if (tystate[k]==cbest) ++count;  
    if ( count > times )  
      break;  
  }  
  if ( count > times ) {  
    best = cbest;  
    times = count; j++;  
  }  
}
```

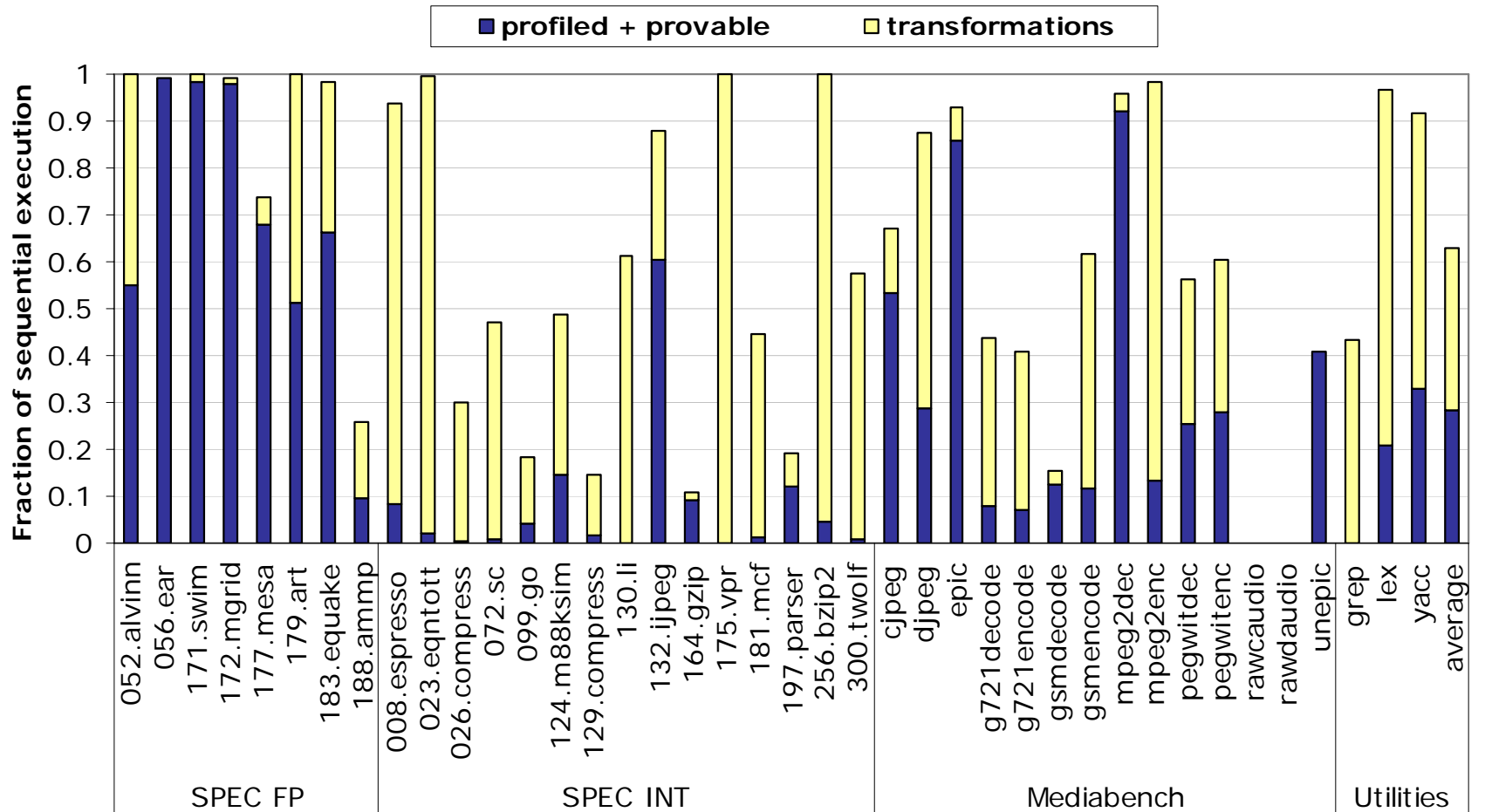
1 %



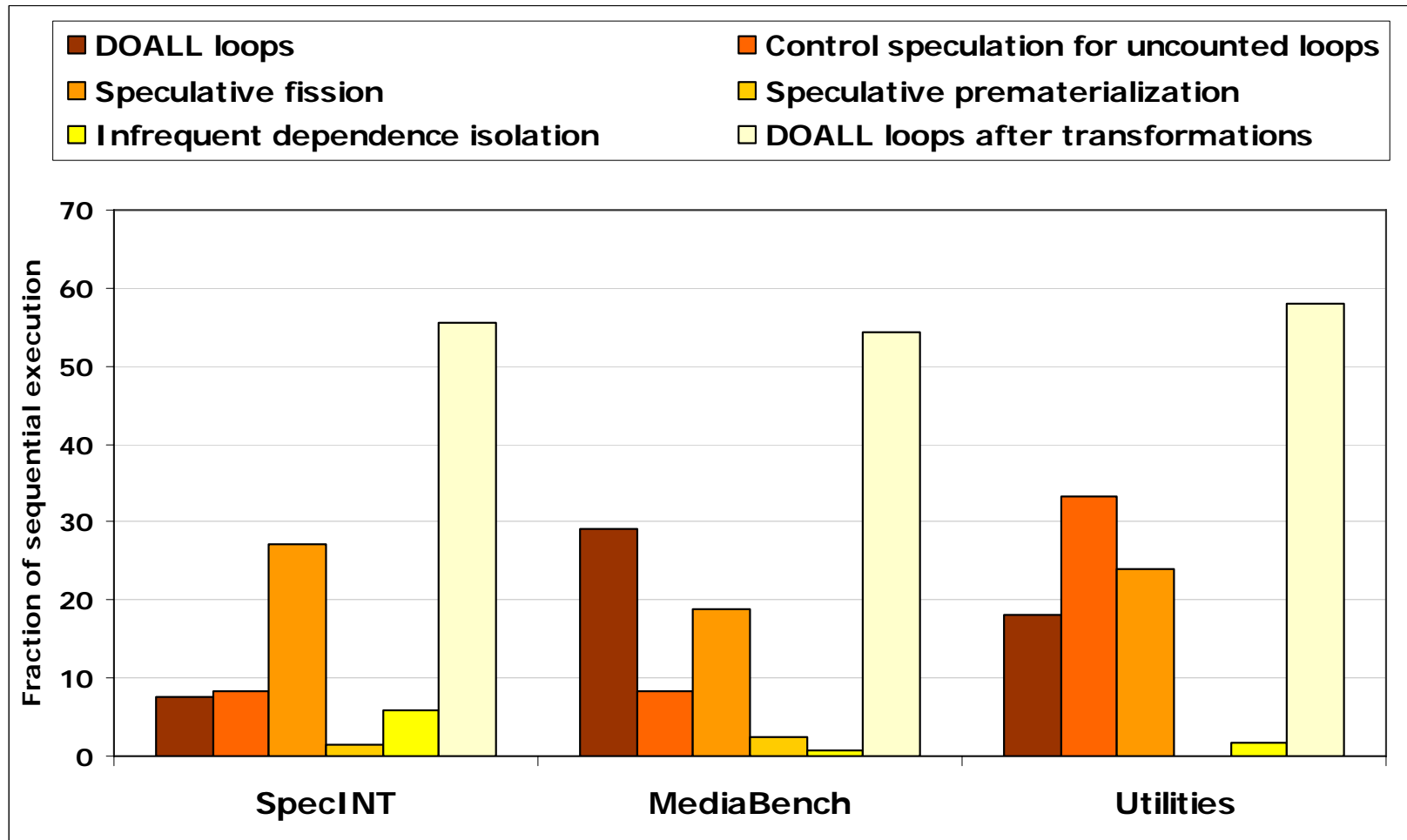
DOALL Coverage – Profiled and Transformed



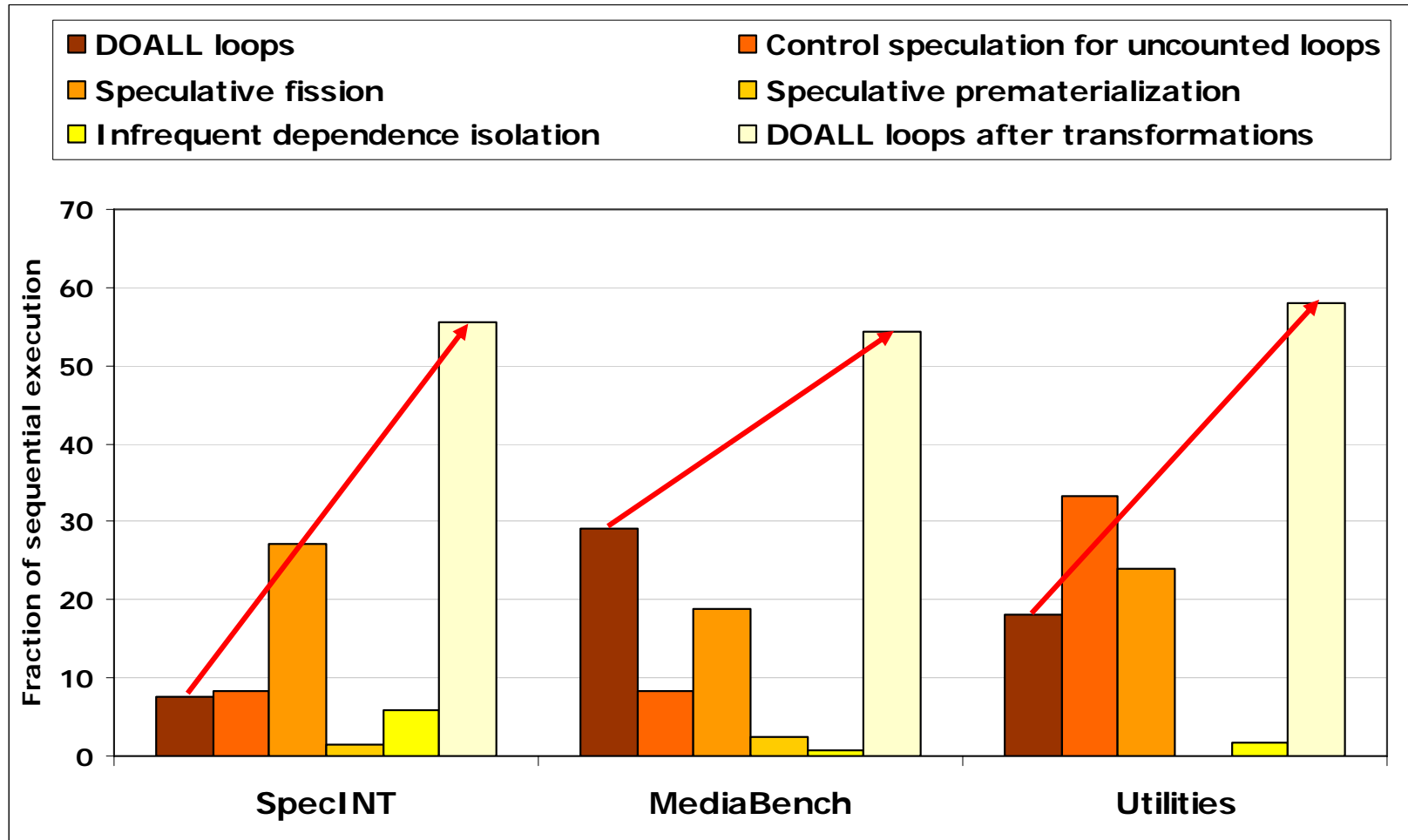
DOALL Coverage – Profiled and Transformed



Coverage Breakdown



Coverage Breakdown

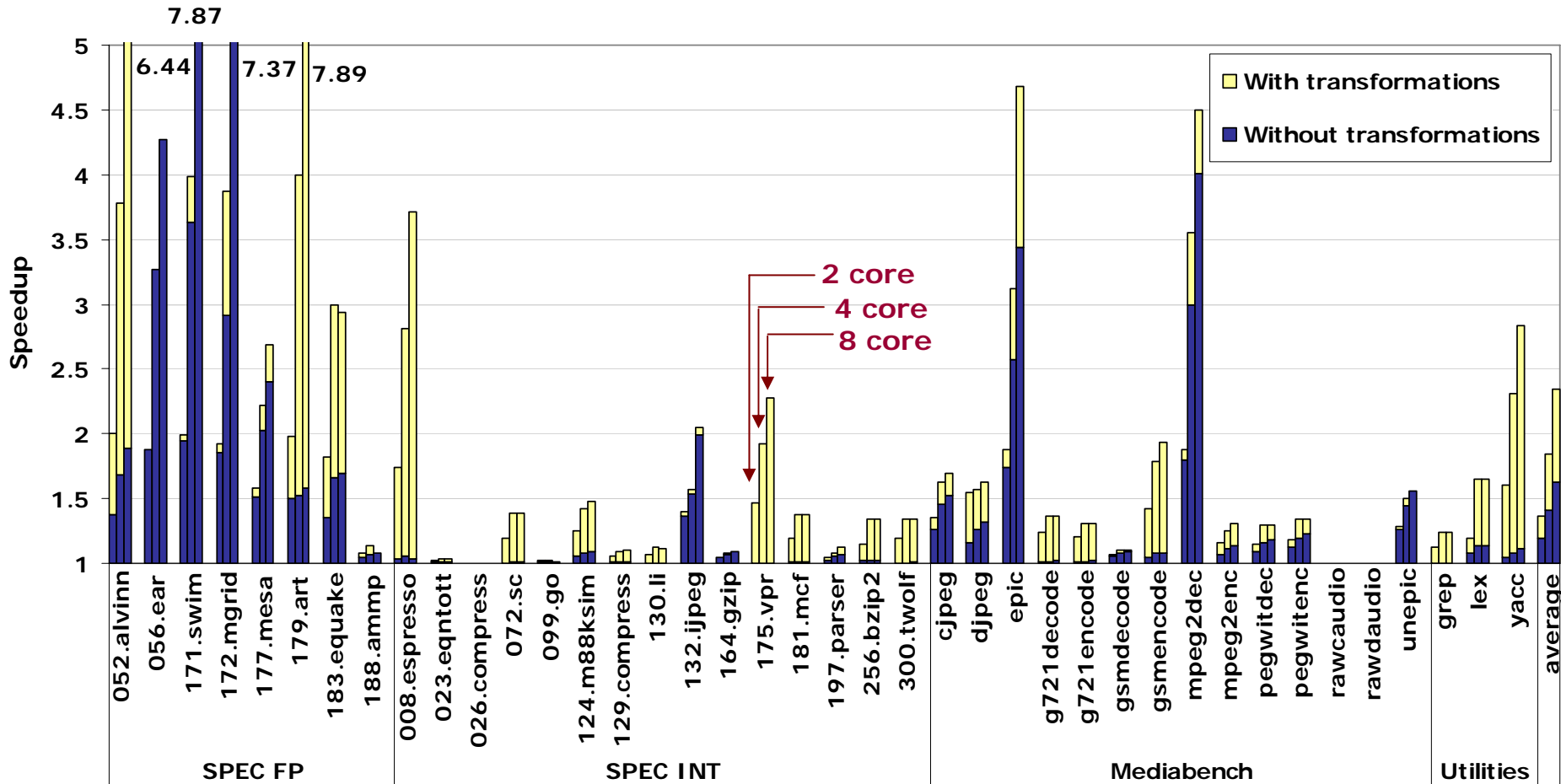


Experimental Setup

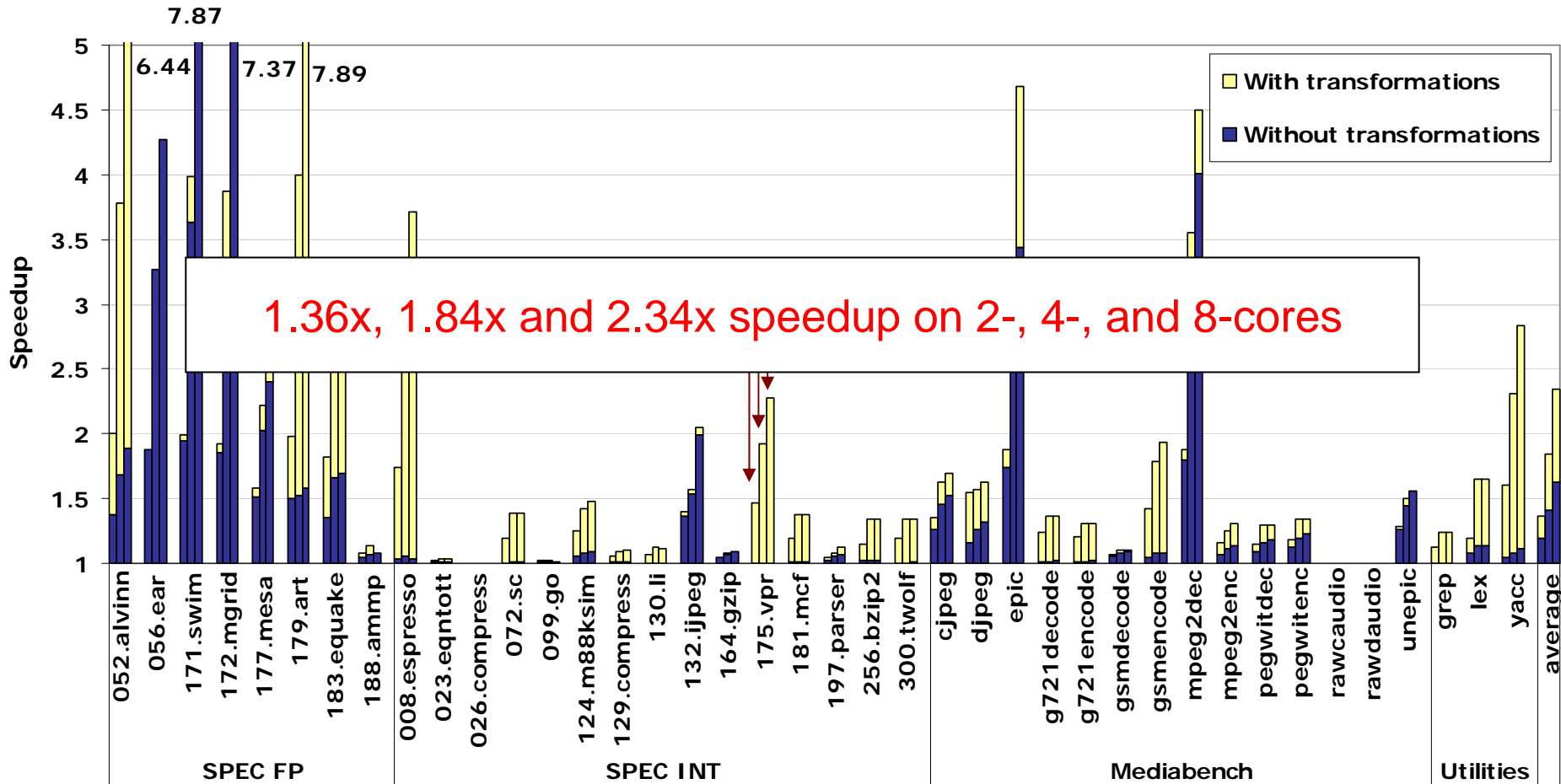
- **OpenIMPACT compiler**
- **Multicore simulator**
 - Simulates up to 8 ARM9-like processors
 - Models scalar operand network
 - Assumes perfect memory system
 - Uses STM library to emulate HTM functionality



Speedup



Speedup



Conclusion

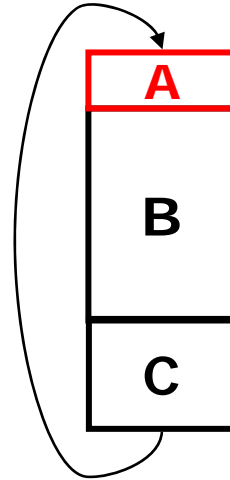
- Figure out ways to use available resources for legacy applications
 - Codes like error handlers, linked list & tree traversal limit parallelism
- Compiler analysis and optimization looks promising
- 1.84x speedup on 4 cores after transformations compared to 1.41x



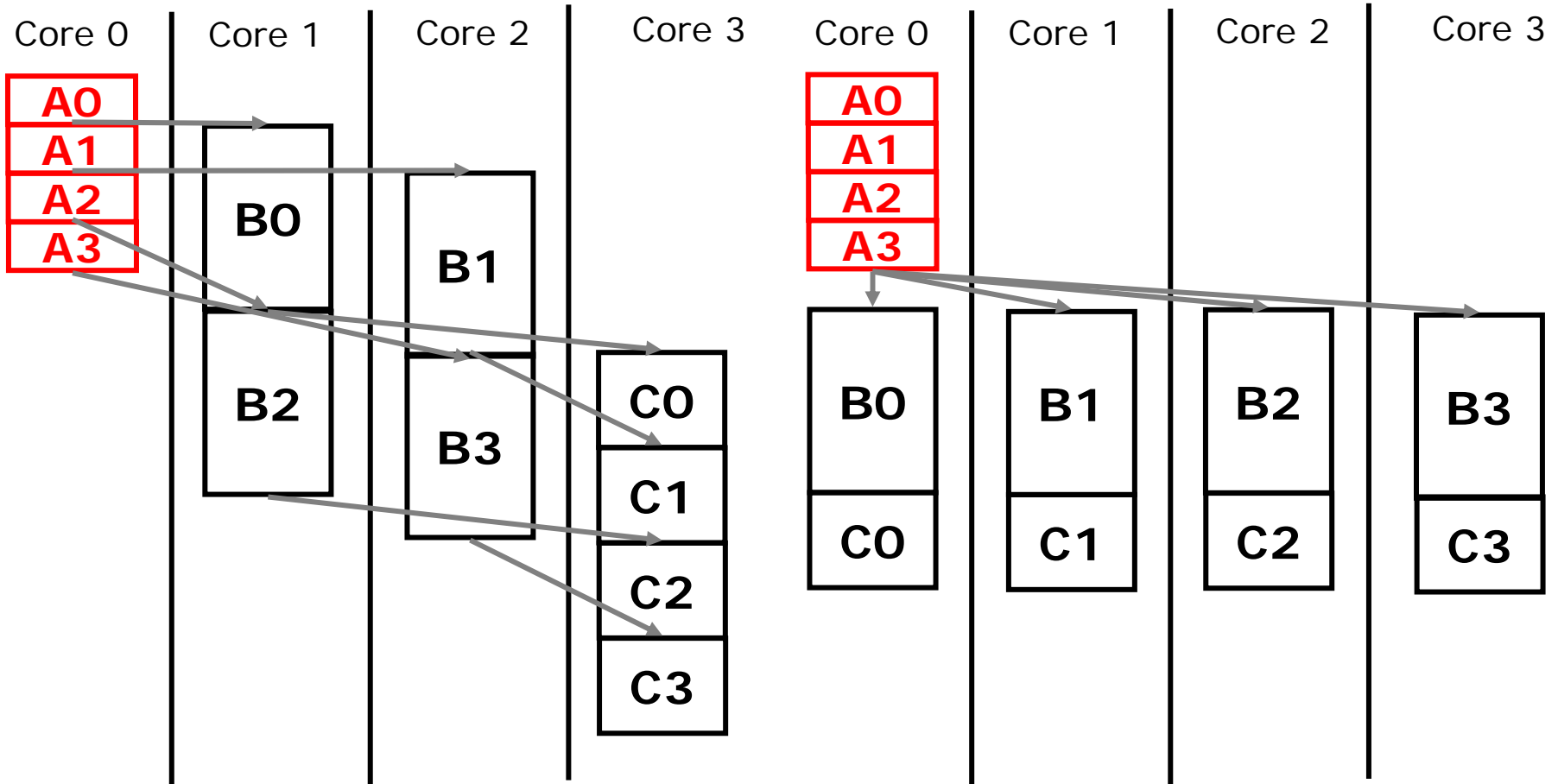
Questions?

Thank you!

SpecDSWP vs. Speculative Fission



SpecDSWP vs. Speculative Fission



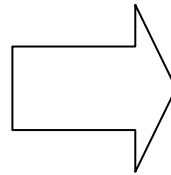
Speculative Prematerialization

```
for (...) {  
1:  current = ...;  
2:  work(last);  
3:  last = current;  
}
```



Speculative Prematerialization

```
for (...) {  
1:  current = ...;  
2:  work(last);  
3:  last = current;  
}
```



```
XBEGIN  
1':  current =  
3':  last =  
      for (...) {  
1:    current = ...;  
2:    work(last);  
3:    last = current;  
      }  
XCOMMIT
```

