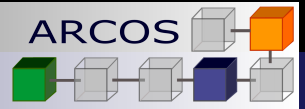
Explotación del paralelismo a nivel de instrucción

Arquitectura de Computadores

J. Daniel García Sánchez (coordinador)
David Expósito Singh
Javier García Blas
Óscar Pérez Alonso
J. Manuel Pérez Lobato

Grupo ARCOS
Departamento de Informática
Universidad Carlos III de Madrid



Aprovechamiento de ILP

- ILP directamente aplicables a bloques básicos.
  - **Bloque básico**: secuencia de instrucciones sin saltos.
  - Programa **típico** en MIPS:
    - Tamaño medio de bloque básico de 3 a 6 instrucciones.
    - Poco aprovechamiento de ILP dentro del bloque.
  - Es necesario explotar ILP entre bloques básicos.

Ejemplo

```
for (i=0;i<1000;i++) {
  x[i] = x[i] + y[i];
}
```

- **Paralelismo a nivel de bucle**.
  - Transformable a ILP.
  - Por compilador o hardware.
- **Alternativa**:
  - Instrucciones vectoriales.
  - Instrucciones SIMD en procesador.

Planificación y desenrollamiento de bucles

- **Aprovechamiento de paralelismo**:
  - Entrelazar ejecución de instrucciones no relacionadas.
  - Rellenar detenciones con instrucciones.
  - No alterar los efectos del programa original.

- El compilador puede usar el conocimiento detallado de la arquitectura.

Aprovechamiento de ILP

Ejemplo

```
for (i=999;i>=0;i--) {
  x[i] = x[i] + s;
}
```

- Cuerpo de cada iteración es independiente.

Latencias entre instrucciones

Instrucción que produce resultado	Instrucción que usa resultado	Latencia (ciclos)
Operación ALU FP	Operación ALU FP	3
Operación ALU FP	Almacenar doble	2
Cargar doble	Operación ALU FP	1
Cargar doble	Almacenar doble	0

Código compilado

- **R1** → Último elemento de array.
- **F2** → Escalar **s**.
- **R2** → Precalculado para que **8(R2)** sea primer elemento de array.

Código ensamblador

```
Bucle:  L.D  F0, 0(R1)          ; F0 <- x[i]
        ADD.D F4, F0, F2        ; F4 <- F0 + s
        S.D  F4, 0(R1)          ; x[i] <- F4
        DADDUI R1, R1, #-8      ; i--
        BNE R1, R2, Bucle       ; Bifurcar Si R1!=R2
```

Detenciones en la ejecución

Original

```
Bucle:  L.D F0, 0(R1)
        ADD.D F4, F0, F2
        S.D F4, 0(R1)
        DADDUI R1, R1, #-8
        BNE R1, R2, Bucle
```

Detenciones

```
Bucle:  L.D F0, 0(R1)
        <stall>
        ADD.D F4, F0, F2
        <stall>
        <stall>
        S.D F4, 0(R1)
        DADDUI R1, R1, #-8
        <stall>
        BNE R1, R2, Bucle
```

Planificación de bucle

Original

```
Bucle:  L.D F0, 0(R1)
        <stall>
        ADD.D F4, F0, F2
        <stall>
        <stall>
        S.D F4, 0(R1)
        DADDUI R1, R1, #-8
        <stall>
        BNE R1, R2, Bucle
```

- 9 ciclos por iteración.

Planificado

```
Bucle:  L.D F0, 0(R1)
        DADDUI R1, R1, #-8
        ADD.D F4, F0, F2
        <stall>
        <stall>
        S.D F4, 8(R1)
        BNE R1, R2, Bucle
```

- 7 ciclos por iteración.

Desenrollamiento de bucles

- **Idea**:
  - Replicar varias veces el cuerpo del bucle.
  - Ajustar el código de terminación.
  - Usa registros distintos para cada réplica para reducir dependencias.

- **Efecto**:
  - Aumenta la longitud del bloque básico.
  - Incrementa el aprovechamiento ILP disponible.

Desenrollamiento

Desenrollado (x4)

```
Bucle:  L.D F0, 0(R1)
        ADD.D F4, F0, F2
        S.D F4, 0(R1)
        L.D F6, -8(R1)
        ADD.D F8, F6, F2
        S.D F8, -8(R1)
        L.D F10, -16(R1)
```

Desenrollado (x4)

```
        ADD.D F12, F10, F2
        S.D F12, -16(R1)
        L.D F14, -24(R1)
        ADD.D F16, F14, F2
        S.D F16, -24(R1)
        DADDUI R1, R1, #-32
        BNE R1, R2, Bucle
```

- 4 iteraciones requieren más registros.
- Este ejemplo asume tamaño de array múltiplo de 4.

Detenciones y desenrollamiento

Desenrollado (x4)

```
Bucle:  L.D F0, 0(R1)
        <stall>
        ADD.D F4, F0, F2
        <stall>
        <stall>
        S.D F4, 0(R1)
        L.D F6, -8(R1)
        <stall>
        ADD.D F8, F6, F2
        <stall>
        <stall>
        S.D F8, -8(R1)
        L.D F10, -16(R1)
        <stall>
```

Desenrollado (x4)

```
        ADD.D F12, F10, F2
        <stall>
        <stall>
        S.D F12, -16(R1)
        L.D F14, -24(R1)
        <stall>
        ADD.D F16, F14, F2
        <stall>
        <stall>
        S.D F16, -24(R1)
        DADDUI R1, R1, #-32
        <stall>
        BNE R1, R2, Bucle
```

- 27 ciclos por 4 iteraciones $\rightarrow$ 6.75 ciclos por iteración.

Planificación y desenrollamiento

Desenrollado (x4)

```
Bucle:  L.D F0, 0(R1)
        L.D F6, -8(R1)
        L.D F10, -16(R1)
        L.D F14, -24(R1)
        ADD.D F4, F0, F2
        ADD.D F8, F6, F2
        ADD.D F12, F10, F2
        ADD.D F16, F14, F2
        S.D F4, 0(R1)
        S.D F8, -8(R1)
        S.D F12, -16(R1)
        DADDUI R1, R1, #-32
        S.D F16, 8(R1)
        BNE R1, R2, Bucle
```

- Reorganización de código.
  - Respetando dependencias.
  - Semánticamente equivalente.
  - **Objetivo**: Aprovechar *stalls*.
- Actualización de **R1** suficientemente separada de **BNE**.
- 14 ciclos por 4 iteraciones $\rightarrow$ 3.5 ciclos por iteración.

Límites del desenrollamiento de bucles

- **Decremento** de la **ganancia** con cada desenrollamiento.
  - Ganancia limitada a eliminación de detenciones.
  - Sobrecoste se reparte entre iteraciones.
- **Crecimiento** en **tamaño** de código.
  - Puede afectar a la tasa de fallos de la caché de instrucciones.
- **Presión** sobre **banco registros**.
  - Puede generar falta de registros.
  - Si no hay registros suficientes se pierden las ventajas.

Predicción de saltos

- Las **bifurcaciones** tienen un alto impacto sobre el rendimiento de los programas.

- Para **reducir** el impacto:
  - Desenrollamiento de bucles.
  - Predicción de saltos.
    - Tiempo de compilación.
    - Comportamiento de cada bifurcación de forma aislada.
  - Predicción avanzada de saltos:
    - Predictores con correlación.
    - Predictores por turnos.

Planificación dinámica

- El hardware **reordena** la ejecución de instrucciones para **reducir** detenciones manteniendo el comportamiento de flujo de datos y excepciones.
- Capaz de tratar casos no conocidos en tiempo de compilación:
  - Fallos/aciertos en caché.
- Código menos dependiente de un pipeline concreto.
  - Simplifica el compilador.
- Permite la **especulación hardware**.

Predicción con correlación

- Si la primera y la segunda bifurcación se toman, la tercer NO se toma.

Ejemplo

```
if (a==2) { a=0; }
if (b==2) { b=0; }
if (a!=b) { f(); }
```

  - Se mantiene la **historia** de las últimas bifurcaciones para seleccionar entre varios predictores.
  - Un predictor (m, n):
    - Usa el resultado de las m últimas bifurcaciones para seleccionar entre 2^m predictores.
    - Cada predictor de n bits.
  - Predictor $(1, 2)$:
    - Resultado de última bifurcación para elegir entre 2 predictores.

Tamaño del predictor

- Un predictor (m,n) tiene varias entradas por cada dirección de bifurcación.
- Tamaño total:

$$T = 2^m \times n \times entradas_{\text{dirección}}$$

- Ejemplos:
  - $(0, 2)$ con 4K entradas $\rightarrow$ 8 Kb
  - $(2, 2)$ con 4K entradas $\rightarrow$ 32 Kb
  - $(2, 2)$ con 1K entradas $\rightarrow$ 8 Kb

Tasa de fallos

- Predictor con correlación tiene menos fallos que predictor simple de mismo tamaño.

- Predictor con correlación tiene menos fallos que predictor simple con número ilimitado de entradas.

Predicción por turnos

- **Combina dos predictores**:
  - Predictor basado en **información global**.
  - Predictor basado en **información local**.
- Utiliza un selector para elegir entre predictores.
  - El cambio entre dos selecciones usa un contador con saturación (2 bits).
- **Ventaja**:
  - Permite comportamiento distinto para enteros y FP.
- **SPEC**:
  - **Benchmarks enteros** $\rightarrow$ predictor global 40%
  - **Benchmarks FP** $\rightarrow$ predictor global 15%.
- **Usos**: Alpha y AMD Opteron.

Intel Core i7

- Predictor de **dos niveles**:
  - Predictor de primer nivel más pequeño.
  - Segundo predictor más grande como *backup*.
- Cada predictor combina **3 predictores**:
  - Predictor simple de 2 bits.
  - Predictor de historia global.
  - Predictor de salida de bucle (contador de iteraciones).
- Además:
  - Predicción de **saltos indirectos**.
  - Predicción de **direcciones de retorno**.

Planificación dinámica

- **Idea**: El hardware reordena la ejecución de instrucciones para reducir detenciones.

- **Ventajas**:
  - Código compilado optimizado para un pipeline ejecuta eficientemente en otro pipeline.
  - Gestiona dependencias desconocidas en tiempo de compilación.
  - Permite tolerar retrasos no predecibles (ej. Fallo de caché).

- **Desventaja**:
  - Mayor complejidad del hardware.

Planificación dinámica

- **Efectos**:
  - Ejecución fuera de orden (OOO).
  - Finalización fuera de orden.
  - Puede introducir **riesgos** WAR y WAW.
- Separación de etapa **ID** en **dos etapas**:
  - **Emisión**: Decodifica la instrucción y comprueba riesgos estructurales.
  - **Lectura de operandos**: Espera hasta que no haya riesgos de datos y lee operandos.
- **Etapa de captación** (**IF**):
  - Capta en registro de instrucciones o cola de instrucciones.

Técnicas de planificación dinámica

- **Scoreboard**:
  - Detiene instrucciones emitidas hasta que hay recursos suficientes y no hay riesgos de datos.
  - Ejemplos: CDC 6600, ARM A8.

- **Algoritmo de Tomasulo**:
  - Elimina dependencias WAR y WAW con renombrado de registros.
  - Ejemplos: IBM 360, Intel Core i7.

Bifurcaciones y límites de paralelismo

- Al aumentar el paralelismo conseguido, las **dependencias de control** se convierten en problema.
  - La predicción de bifurcaciones no es suficiente.

- El siguiente paso es la **especulación** sobre el **resultado de las bifurcaciones** y la **ejecución** asumiendo que la **especulación fue correcta**.
  - Se capta, emite y ejecuta instrucciones.
  - Se necesita un mecanismo de tratamiento si la especulación no era correcta.

Componentes

- **Ideas**:
  - **Predicción dinámica de saltos**: Selecciona las instrucciones a ejecutar.
  - **Especulación**: Ejecución antes de que se resuelvan dependencias de control y capacidad para deshacer.
  - **Planificación dinámica**.
- Para conseguirlo se debe **separar**:
  - El **paso del resultado** de una instrucción a otra que lo usa.
  - La **finalización** de la instrucción.
- **IMPORTANTE**:No se actualiza el estado del procesador (registros/memoria) hasta que no se tiene confirmación.

Solución

- **Reorder Buffer** (**ROB**):
  - Cuando se finaliza una instrucción se escribe en el **ROB**.
  - Cuando se confirma su ejecución se escribe en destino real.
  - Las instrucciones leen datos modificados del **ROB**.

- Entradas del **ROB**:
  - **Tipo de instrucción**: branch, store, operación de registro.
  - **Destino**: Id de registro o dirección de memoria.
  - **Valor**: Valor del resultado de la instrucción.
  - **Ready**: Indica si la instrucción se ha completado.

CPI $<$ 1

- CPI $\geq$ 1 $\rightarrow$ Emisión de una instrucción por ciclo.
- Procesadores de múltiple emisión (**CPI** $<$ 1 $\rightarrow$ **IPC** $>$ 1):
  - Procesadores superescalares planificados **estáticamente**.
    - Ejecución en orden.
    - Número variable de instrucciones por ciclo.
  - Procesadores superescalares planificados **dinámicamente**.
    - Ejecución fuera de orden.
    - Número variable de instrucciones por ciclo.
  - Procesadores **VLIW** (*Very Long Instruction Word*).
    - Varias instrucciones empaquetadas en paquete.
    - Planificación estática.
    - Paralelismo explícito a nivel de instrucción por el compilador.

Enfoques de la emisión múltiple

- Varios enfoques posibles con emisión múltiple.
  - **Superescalar estático**.
  - **Superescalar dinámico**.
  - **Superescalar especulativo**.
  - **VLIW/LIW**.
  - **EPIC**.

Superescalar estático

- **Emisión**: Dinámica.
- **Detección de riesgos**: Hardware.
- **Planificación**: Estática.
- **Propiedad discriminante**:
  - Ejecución en orden.

- **Ejemplos**:
  - MIPS.
  - ARM Cortex-A7.

Superescalar dinámico

- **Emisión**: Dinámica.
- **Detección de riesgos**: Hardware.
- **Planificación**: Dinámica.
- **Propiedad discriminante**:
  - Ejecución fuera de orden sin especulación.

- **Ejemplos**: Ninguno.

Superescalar especulativo

- **Emisión**: Dinámica.
- **Detección de riesgos**: Hardware.
- **Planificación**: Dinámica con especulación.
- **Propiedad discriminante**:
  - Ejecución fuera de orden con especulación.

- **Ejemplos**:
  - Intel Core i3, i5, i7.
  - AMD Phenom.
  - IBM Power 7

VLIW

- Empaquetado de varias operaciones en una instrucción.
- Instrucción ejemplo en ISA VLIW:
  - Una instrucción entera o una bifurcación.
  - Dos operaciones de coma flotante independientes.
  - Dos referencias a memoria independientes.
- **IMPORTANTE**: El código debe presentar suficiente paralelismo.

VLIW / LIW

- **Emisión**: Estática.
- **Detección de riesgos**: Principalmente software.
- **Planificación**: Estática.
- **Propiedad discriminante**:
  - Todos los riesgos determinados e indicados por el compilador.

- **Ejemplos**:
  - DSPs (p. ej. TI C6x).

Problemas de VLIW

- **Desventajas** del modelo original **VLIW**:
  - Complejidad de encontrar paralelismo estáticamente.
  - Tamaño del código generado.
  - No tienen hardware de detección de riesgos.
  - Problemas de compatibilidad binaria mayores que en superescalares.

- **EPIC** intenta resolver la mayoría de estos problemas.

EPIC

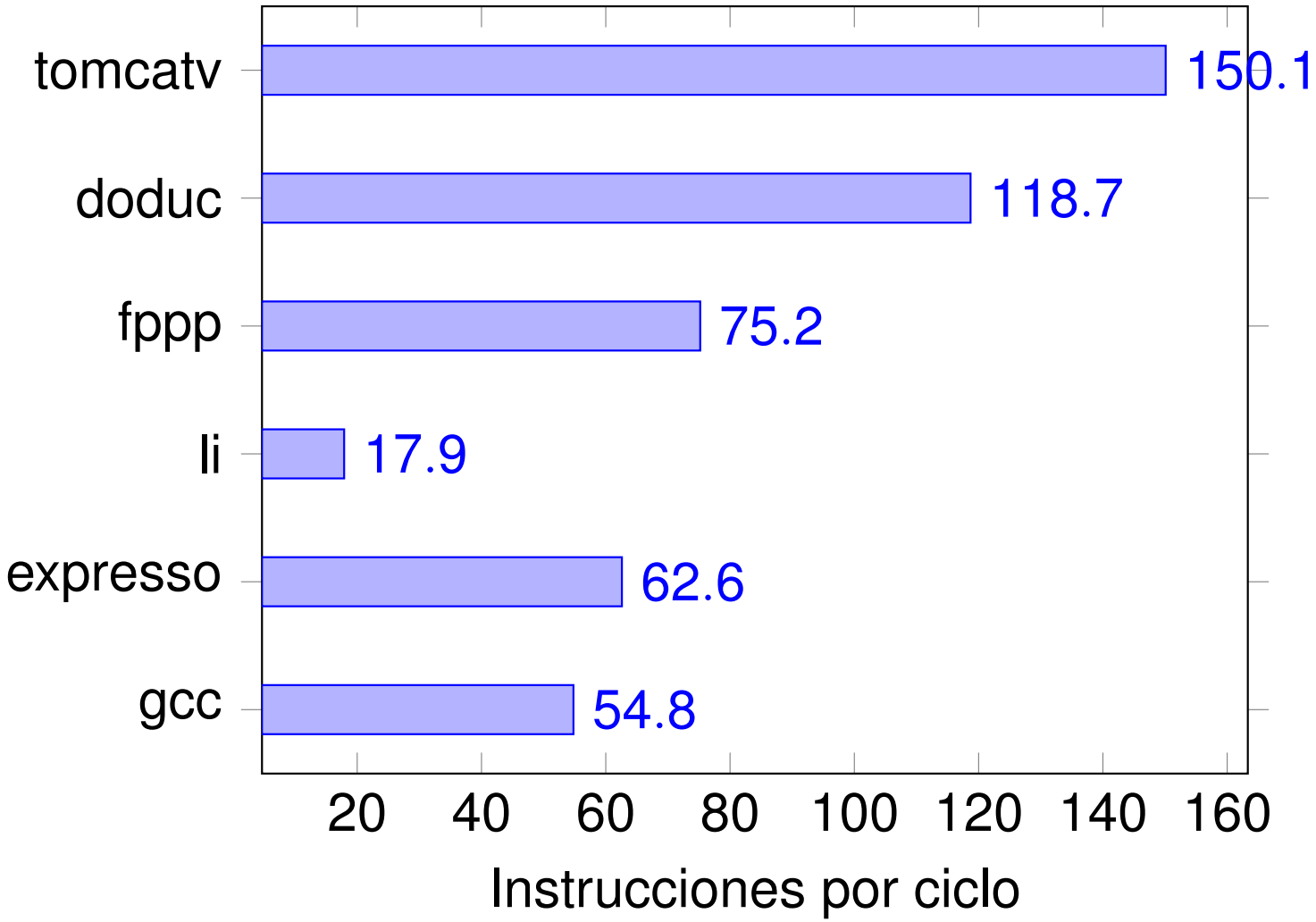
- **Emisión**: Principalmente estática.
- **Detección de riesgos**: Principalmente software.
- **Planificación**: Sobre todo estática.
- **Propiedad discriminante**:
  - Todos los riesgos determinados e indicados por el compilador.

- **Ejemplos**:
  - Itanium.

Límites de ILP

- Para estudiar el **ILP** máximo se modela el **procesador ideal**.
- **Procesador ideal**:
  - **Renombrado de registros infinito**: Se pueden evitar todos los riesgos WAR y WAW.
  - **Predicción de bifurcación perfecta**: Todos las predicciones de bifurcaciones condicionales dan acierto.
  - **Predicción perfecta de saltos**: Todos los saltos (incluyendo retornos) se predicen correctamente.
  - **Análisis perfecto de alias de direcciones de memoria**: Se puede mover un load antes de un store si la dirección no es idéntica.
  - **Cachés perfectas**: Todos los acceso a caché requieren un ciclo de reloj (siempre hay acierto).

ILP disponible



Sin embargo . . .

- Más ILP implica más lógica de control:
  - Cachés de menor tamaño.
  - Ciclos de reloj más largos.
  - Mayor consumo de energía.

- **Limitación práctica**:
  - Emisión de 3 a 6 instrucciones por ciclo.

¿Por qué TLP?

- Algunas aplicaciones presentan más **paralelismo natural** que el que se puede conseguir con **ILP**.
  - Servidores, aplicaciones científicas, . . .

- Emergen **dos modelos**:
  - **Paralelismo a nivel de hilos (TLP)**:
    - **Hilo**: Proceso con sus propias instrucciones y datos.
    - Puede ser parte un programa o programa independiente.
    - Cada hilo tiene asociado su **estado** (instrucciones, datos, PC, registros, . . .).
  - **Paralelismo a nivel de datos (DLP)**:
    - Operaciones idénticas sobre distintos datos.

TLP

- **ILP** explota paralelismo implícito dentro un bloque básico o en bucles.
- **TLP** utiliza múltiples hilos de ejecución que son inherentemente paralelos.
- **Objetivo de TLP**:
  - Utilizar múltiples flujos de instrucciones para mejorar:
    - **Tasa de procesamiento** de computadores que ejecutan muchos programas.
    - **Tiempo de ejecución** de programas multi-hilo.

Ejecución multi-hilo

- Múltiples hilos comparten las unidades funcionales de un procesador solapando su uso.
  - Necesidad de replicar n-veces el estado.
    - Banco de registros, PC, tabla de páginas (si hilos no pertenecen al mismo programa).
    - Memoria compartida mediante mecanismos de memoria virtual.
    - Hardware para cambio de hilo rápido.
  - **Tipos**:
    - **Grano fino**: Cambio de hilo en cada instrucción.
    - **Grano grueso**: Cambio de hilo en detenciones (ej. Fallo de caché).
    - **Multi-hilo simultáneo**: Grano fino con emisión múltiple y planificación dinámica.

Multi-hilo de grano fino

- Se alterna entre hilos en cada instrucción.
  - Se entrelaza la ejecución de los hilos.
  - Normalmente se hace *round-robin*.
  - Se excluyen de *round-robin* hilos en detención (*stall*).
  - El procesador debe poder cambiar de hilo en cada ciclo de reloj.
- **Ventaja**:
  - Puede ocultar detenciones cortas y largas.
- **Desventaja**:
  - Retrasa la ejecución de hilos individuales por reparto.
- **Ejemplo**: Sun Niagara.

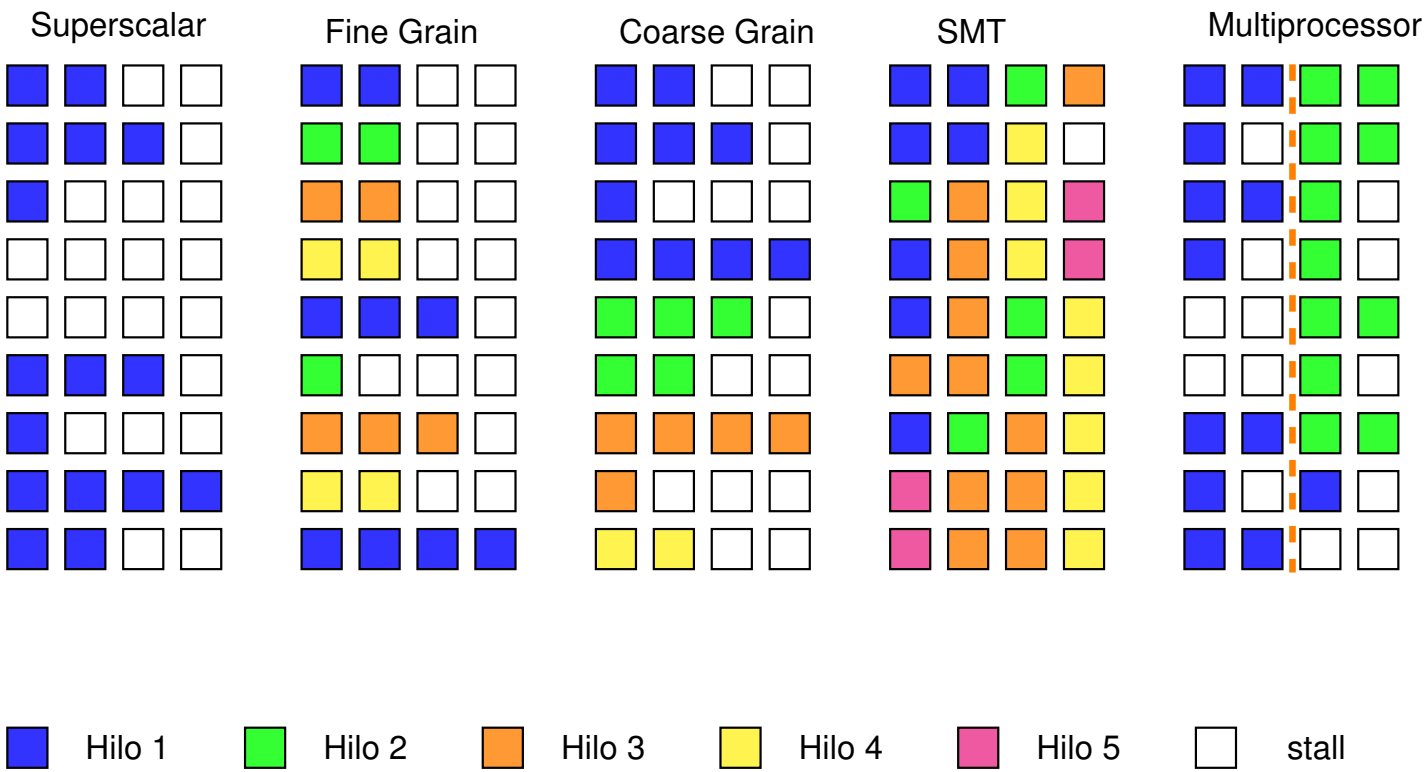
Multi-hilo de grano grueso

- Cambia de hilo solamente en detenciones largas.
  - **Ejemplo**: Fallo en caché L2.
- **Ventajas**:
  - No hace falta cambio de hilo excesivamente rápido.
  - No retrasa hilos individuales.
- **Desventajas**:
  - Se debe vaciar o congelar el pipeline.
  - Se debe llenar el pipeline con instrucciones del nuevo hilo (latencia).
- Apropiado cuando rellenado del pipeline tarda mucho menos que tiempo de detención.
  - **Ejemplo**: IBM AS/400.

SMT: Multi-hilo simultáneo

- **Idea**: Procesadores con planificación dinámica ya tienen muchos mecanismo de soporte para multi-hilo.
  - Grandes conjuntos de registros virtuales.
    - Registros para múltiples hilos.
  - Renombrado de registros.
    - Evita mezcla en acceso a registros de hilos.
  - Finalización fuera de orden.
- **Modificaciones**:
  - Tabla de renombrado por hilo.
  - Registros PC separados.
  - ROB separados.
- **Ejemplos**: Intel Core i7, IBM Power 7

TLP: Resumen


Superscalar
Fine Grain
Coarse Grain
SMT
Multiprocessor
Hilo 1
Hilo 2
Hilo 3
Hilo 4
Hilo 5
stall

Resumen

- El desenrollamiento de bucles permite ocultar las latencias de detenciones, pero da una ganancia limitada.
- La planificación dinámica gestiona detenciones desconocidas en tiempo de compilación.
- Las técnicas especulativas se apoyan de la predicción de saltos y la planificación dinámica.
- La emisión múltiple en ILP queda limitada de forma práctica de 3 a 6.
- SMT como aproximación a TLP dentro un núcleo.

Referencias

- **Computer Architecture. A Quantitative Approach**
  5th Ed.
  Hennessy and Patterson.
  Secciones 3.1, 3.2, 3.3, 3.4, 3.6, 3.7, 3.10, 3.12.

- **Ejercicios recomendados**:
  - 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.11, 3.14, 3.17.

Explotación del paralelismo a nivel de instrucción

Arquitectura de Computadores

J. Daniel García Sánchez (coordinador)
David Expósito Singh
Javier García Blas
Óscar Pérez Alonso
J. Manuel Pérez Lobato

Grupo ARCOS
Departamento de Informática
Universidad Carlos III de Madrid