Optimizing Dynamic Binary Translation for SIMD Instructions

Jianhui Li, Qi Zhang, Shu Xu, Bo Huang

Intel China Software Center

Outlines

- Introduction
- Challenges and opportunity for SIMD instructions translation
- IA32 Execution Layer® background
- SIMD data type tracking algorithm
- Optimizing SIMD data type tracking algorithm
- Performance evaluation
- Conclusion and Acknowledgements

Introduction

- SIMD instruction could be divided into two categories: Scalar type (MOVSS, MULSS), Parallel type (MULPS, MOVAPS)

- Translation for SIMD instructions are becoming critical.
  - Intel MMX, SSE/SSE2/SSE3 , HP MAX(*), IBM AltiVec(*), Sun Sparc VIS(*), etc
  - In SPEC2K compiled by Intel Compiler 8.1, average execution percentage for SSE/SSE2/SSE3 is 57.66%, for parallel SSE/SSE2/SSE3 instruction 10.06%

- A product Binary Translator has to support SIMD translation efficiently

* Other brands and names are the property of their respective owners

Challenges and opportunities for SIMD instruction translation

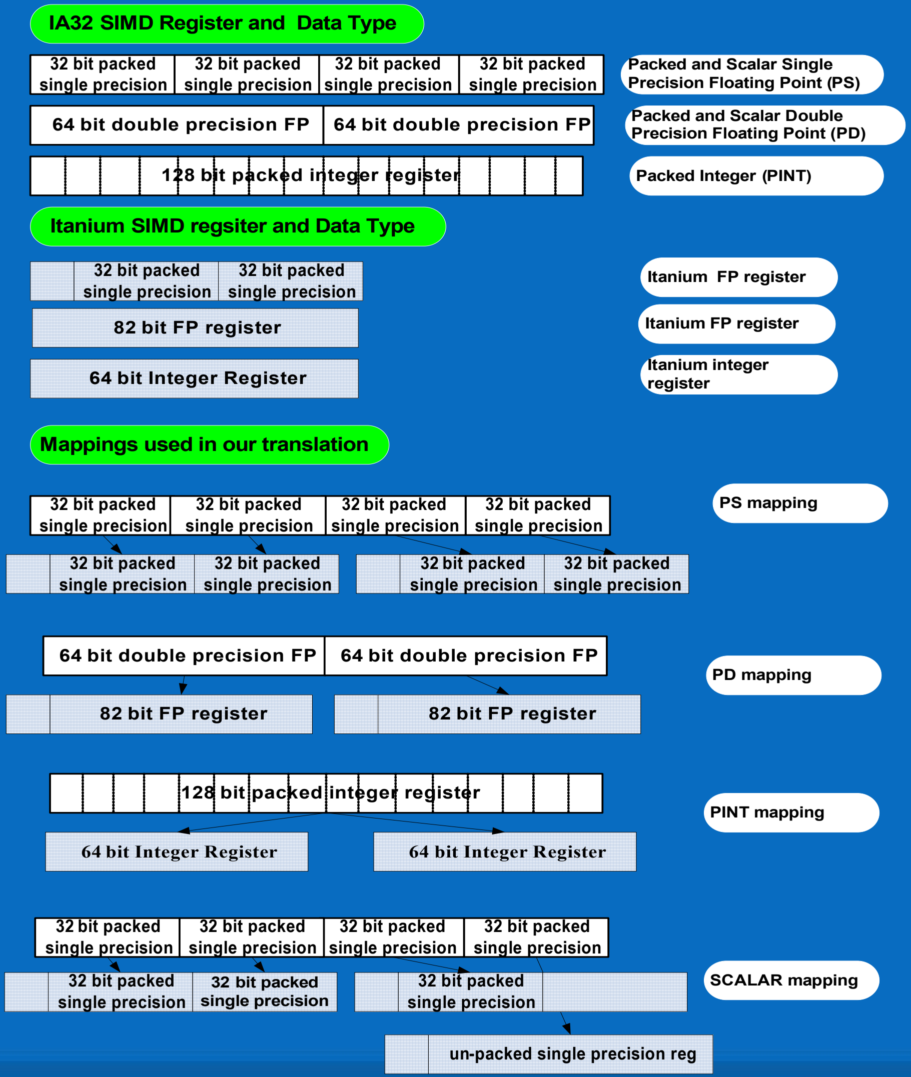
- Canonical register: SIMD register on source architecture; Mapping register: register on target architecture to simulate canonical register

- Probably, more mapping registers are needed to simulate one canonical register if mapping register supports less data type than canonical registers

- One register written by previous SIMD inst is accessed by later SIMD inst in another data type will cause data type mismatch

- Synchronization code is needed when data type mismatch happens (performance challenge)

- Different SIMD instructions may have the same effect and can be interchanged (performance opportunities)



IA32 SIMD Register and Data Type

32 bit packed single precision | 32 bit packed single precision | 32 bit packed single precision | 32 bit packed single precision — Packed and Scalar Single Precision Floating Point (PS)

64 bit double precision FP | 64 bit double precision FP — Packed and Scalar Double Precision Floating Point (PD)

128 bit packed integer register — Packed Integer (PINT)

Itanium SIMD regsiter and Data Type

32 bit packed single precision | 32 bit packed single precision — Itanium FP register

82 bit FP register — Itanium FP register

64 bit Integer Register — Itanium integer register

Mappings used in our translation

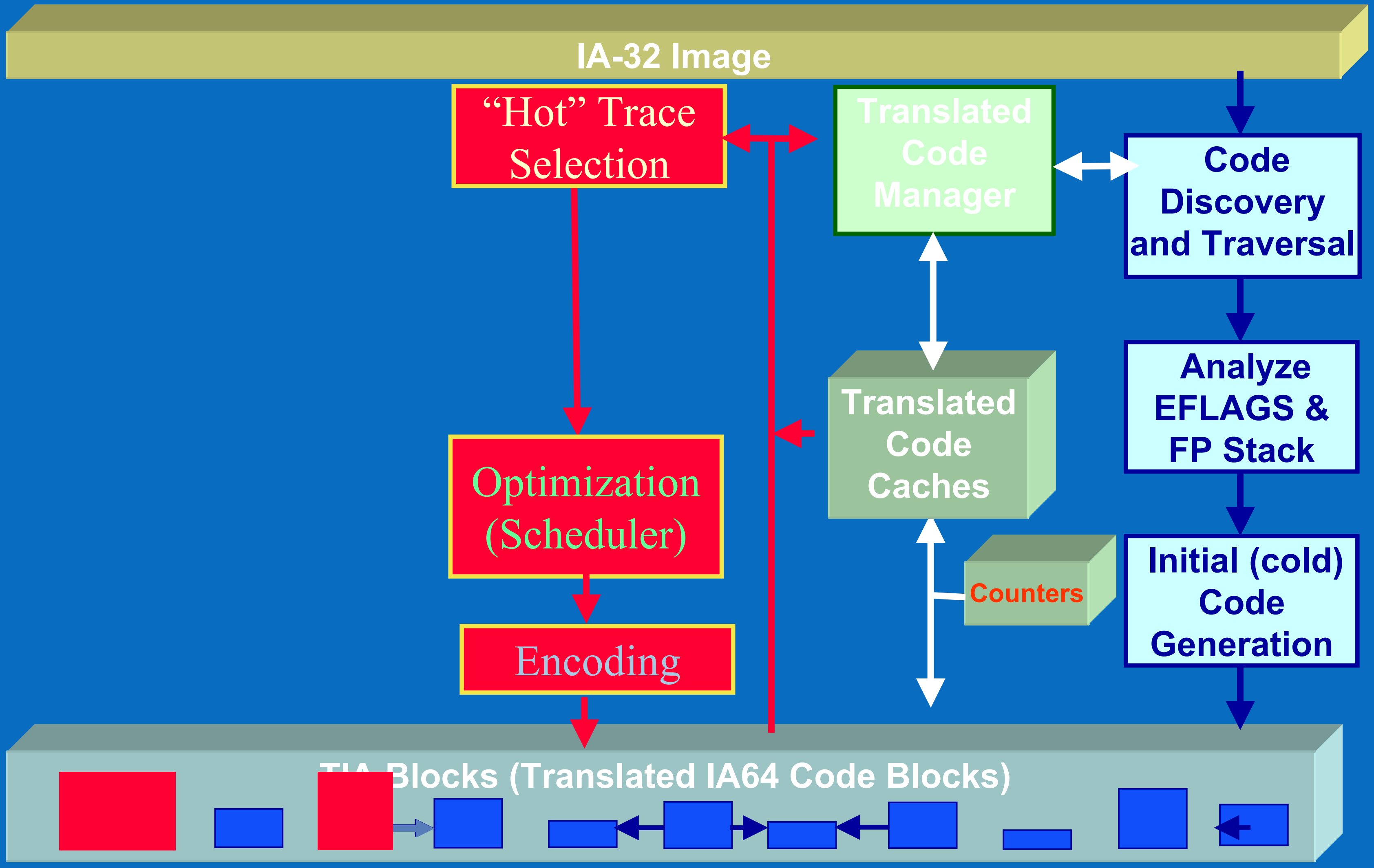
PS mapping: 32 bit packed single precision | 32 bit packed single precision | 32 bit packed single precision | 32 bit packed single precision → 32 bit packed single precision | 32 bit packed single precision; 32 bit packed single precision | 32 bit packed single precision

PD mapping: 64 bit double precision FP | 64 bit double precision FP → 82 bit FP register; 82 bit FP register

PINT mapping: 128 bit packed integer register → 64 bit Integer Register; 64 bit Integer Register

SCALAR mapping: 32 bit packed single precision | 32 bit packed single precision | 32 bit packed single precision | 32 bit packed single precision → 32 bit packed single precision | 32 bit packed single precision; 32 bit packed single precision → un-packed single precision reg

IA32 Execution Layer® background



IA-32 Image
"Hot" Trace Selection
Translated Code Manager
Code Discovery and Traversal
Analyze EFLAGS & FP Stack
Translated Code Caches
Optimization (Scheduler)
Counters
Initial (cold) Code Generation
Encoding
TIA Blocks (Translated IA64 Code Blocks)

Baseline algorithm: SIMD data type tracking algorithm

- Calculating Input/Output SIMD formats for a sequence (Block)

- Detecting and fixing intra-block data type mismatch.

- Detecting and fixing inter-block data type mismatch

Input/Output format for a block

a) Source SSE instruction:

S1: MOVAPD XMM1 xmmword ptr [mem0]

S2: PSUBD XMM1 XMM0

S3: MOVAPD xmmword ptr [mem0] XMM1

S1	XMM0	XMM1
Input	ø	ø
Output	ø	PD

S2	XMM0	XMM1
Input	PINT	PINT
Output	PINT	PINT

S3	XMM0	XMM1
Input	ø	PD
Output	ø	PD

Input and output format for each inst

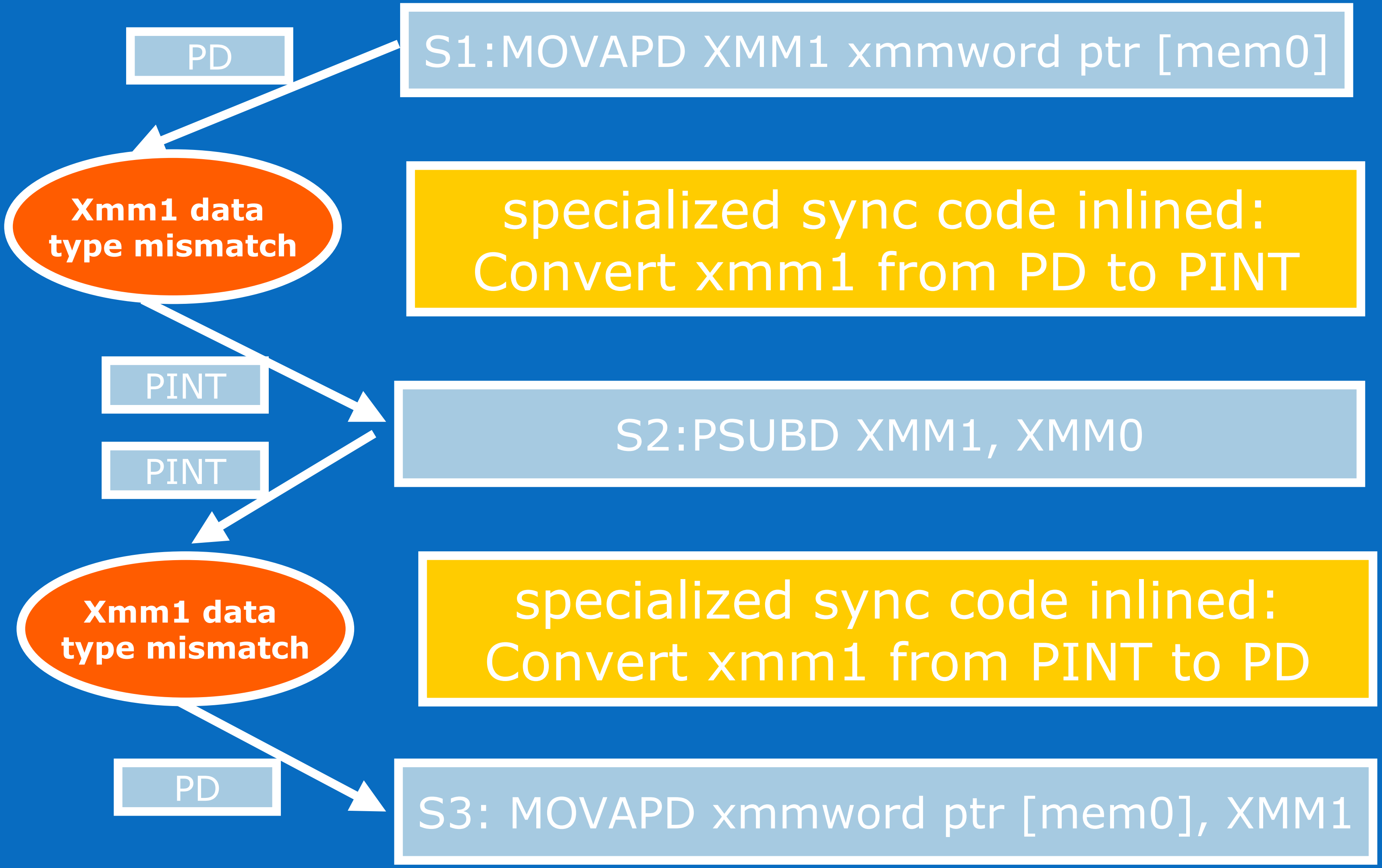
	XMM0	XMM1
Input	PINT	ø
Output	PINT	PINT

Input and output format after merging s1 and s2

	XMM0	XMM1
Input	PINT	ø
Output	PINT	PD

Input and output format of the sequence

Detecting and fixing intra-block mismatch



S1:MOVAPD XMM1 xmmword ptr [mem0]

PD

Xmm1 data type mismatch

specialized sync code inlined: Convert xmm1 from PD to PINT

PINT

S2:PSUBD XMM1, XMM0

PINT

Xmm1 data type mismatch

specialized sync code inlined: Convert xmm1 from PINT to PD

PD

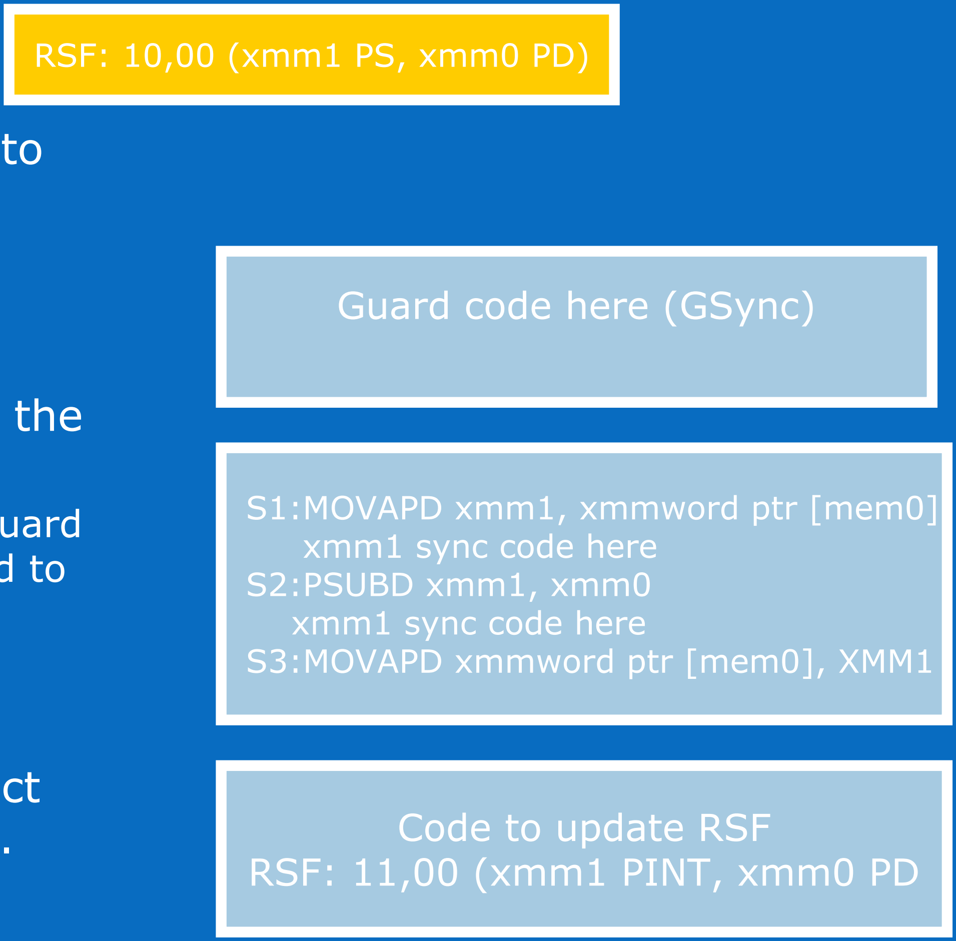
S3: MOVAPD xmmword ptr [mem0], XMM1

Detecting and fixing inter-block mismatch

- A global RSF register is reserved to indicate current canonical register format during runtime

- Some guard code is generated at the beginning of the block
  - If type mismatch is detected by guard code, GSync code will be executed to fix them

- At the exit of one block, code is generated to update RSF to reflect correct canonical register format.



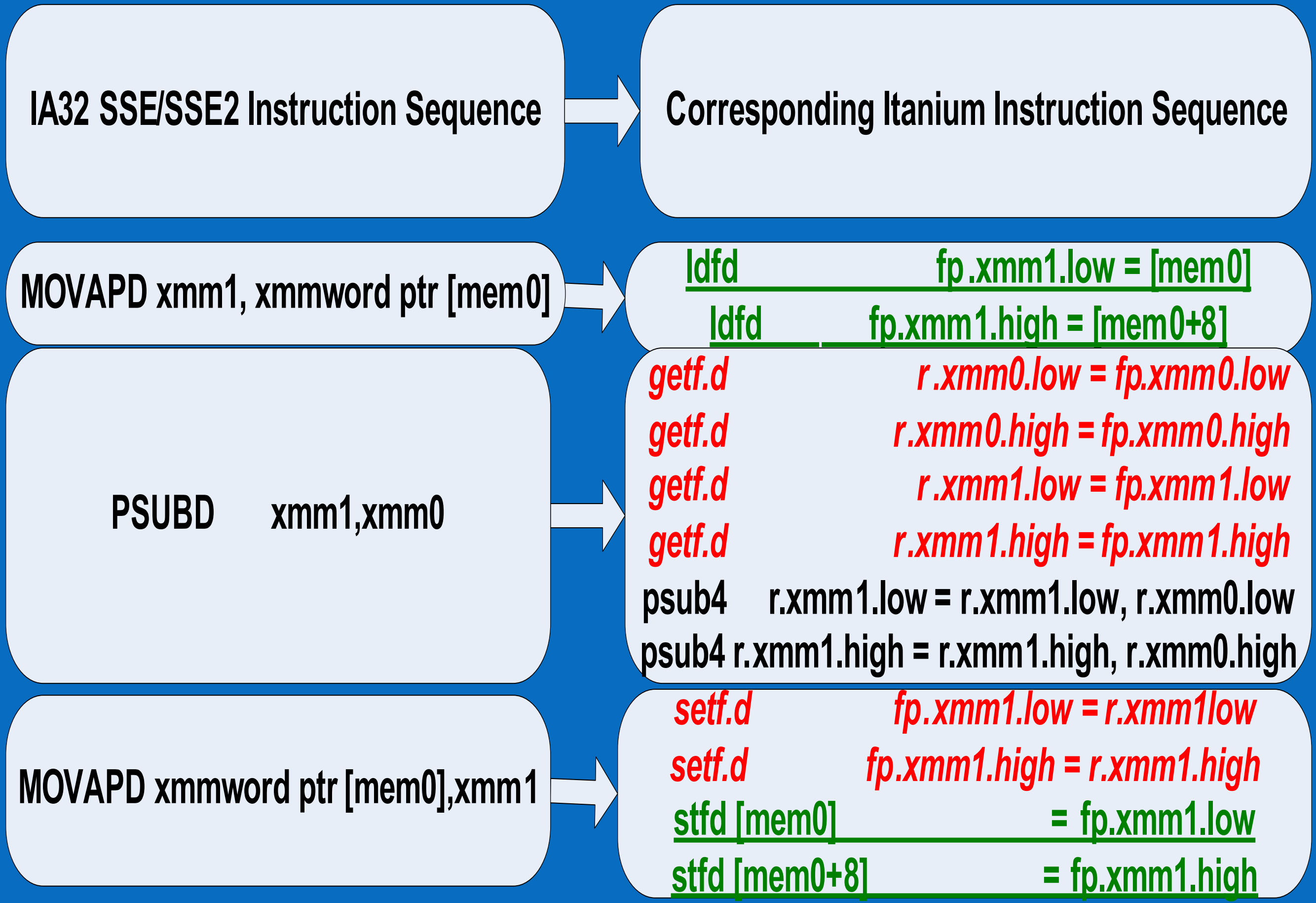
RSF: 10,00 (xmm1 PS, xmm0 PD)

Guard code here (GSync)

```
S1:MOVAPD xmm1, xmmword ptr [mem0]
     xmm1 sync code here
S2:PSUBD xmm1, xmm0
    xmm1 sync code here
S3:MOVAPD xmmword ptr [mem0], XMM1
```

Code to update RSF
RSF: 11,00 (xmm1 PINT, xmm0 PD

Translation result of the example



IA32 SSE/SSE2 Instruction Sequence	Corresponding Itanium Instruction Sequence
MOVAPD xmm1, xmmword ptr [mem0]	ldfd fp.xmm1.low = [mem0] ldfd fp.xmm1.high = [mem0+8]
PSUBD xmm1,xmm0	*getf.d r.xmm0.low = fp.xmm0.low* *getf.d r.xmm0.high = fp.xmm0.high* *getf.d r.xmm1.low = fp.xmm1.low* *getf.d r.xmm1.high = fp.xmm1.high* psub4 r.xmm1.low = r.xmm1.low, r.xmm0.low psub4 r.xmm1.high = r.xmm1.high, r.xmm0.high
MOVAPD xmmword ptr [mem0],xmm1	*setf.d fp.xmm1.low = r.xmm1low* *setf.d fp.xmm1.high = r.xmm1.high* stfd [mem0] = fp.xmm1.low stfd [mem0+8] = fp.xmm1.high

Legend: *Instruction* Conversion code

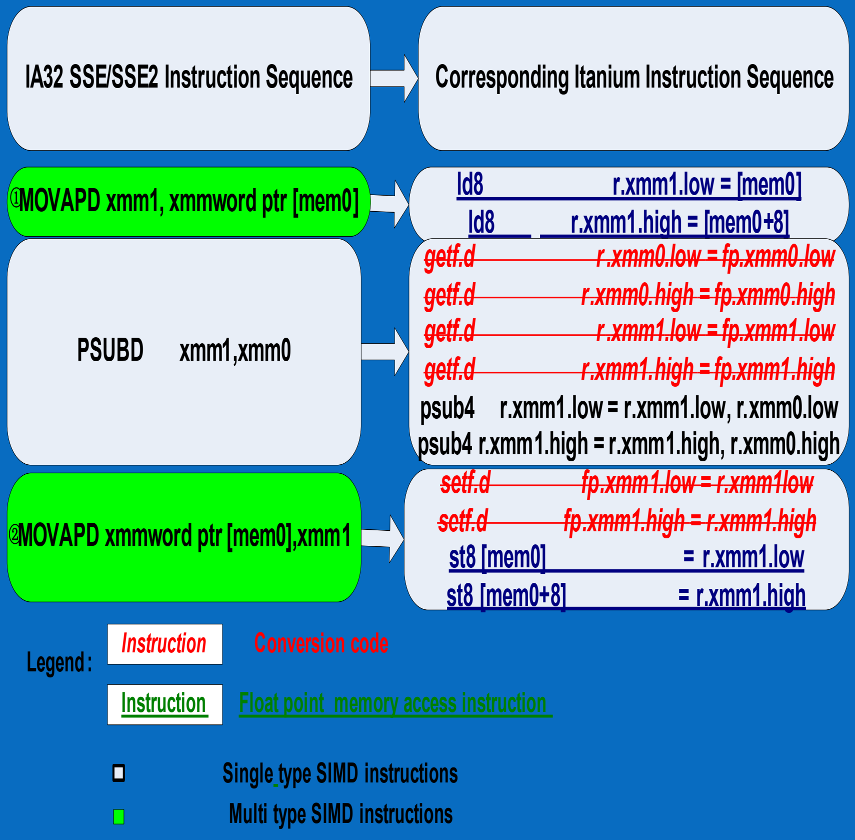
Instruction Float point memory access instruction

Optimization 1: SIMD data type re-assignment

- Utilizing the fact that different SIMD instructions may have the same semantics (multi-type SIMD instruction)

- Generating the lowest cost translation by “data type flow” analysis, which is an extension to the standard data flow analysis

- One cost based heuristics model



IA32 SSE/SSE2 Instruction Sequence → Corresponding Itanium Instruction Sequence

①MOVAPD xmm1, xmmword ptr [mem0] →

ld8 r.xmm1.low = [mem0]
ld8 r.xmm1.high = [mem0+8]

PSUBD xmm1,xmm0 →

~~getf.d r.xmm0.low = fp.xmm0.low~~
~~getf.d r.xmm0.high = fp.xmm0.high~~
~~getf.d r.xmm1.low = fp.xmm1.low~~
~~getf.d r.xmm1.high = fp.xmm1.high~~
psub4 r.xmm1.low = r.xmm1.low, r.xmm0.low
psub4 r.xmm1.high = r.xmm1.high, r.xmm0.high

②MOVAPD xmmword ptr [mem0],xmm1 →

~~setf.d fp.xmm1.low = r.xmm1low~~
~~setf.d fp.xmm1.high = r.xmm1.high~~
st8 [mem0] = r.xmm1.low
st8 [mem0+8] = r.xmm1.high

Legend: *Instruction* Conversion code
Instruction Float point memory access instruction

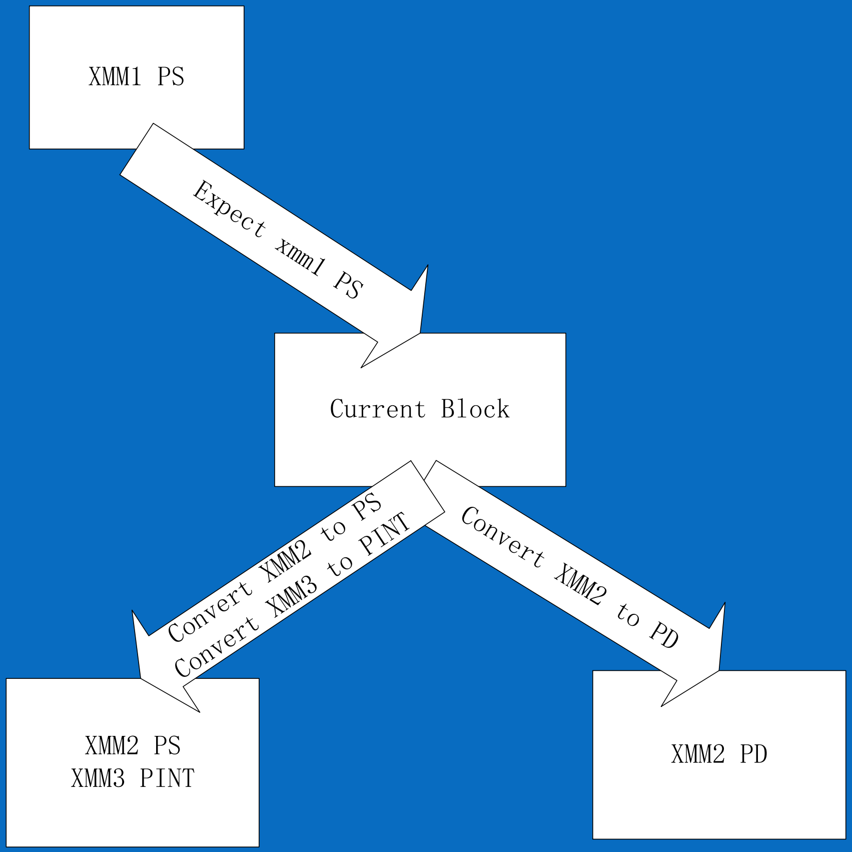
□ Single type SIMD instructions
🟩 Multi type SIMD instructions

Optimization 2: translation time inter-block data type mismatch removal

- If CFG predecessor block's output format is known, take it as current block input format.

- If CFG successor block's input format is known, convert current block's output format consistent with successor block

- If multiple predecessors or successors are encountered, take the hottest one



Optimization 3: runtime inter-block data type mismatch removal

- Do instrumentation in GSyn code.

- Record the execution count and incoming format in one hashtable.

- If execution count reaches one tunable threshold, one piece of specialized data type synchronization code is generated, which only converted the incoming format to current block's expected format.

- The specialized synchronization code is executed first (by code patching), if it fails to handle the incoming format, GSyn will be executed again.

Performance Evaluation

- Using SPEC2K compiled using Intel C/C++ 8.1 (Windows version)
- SSE3 optimization enabled
- Measured on Itanium2 machine (4 CPUs, 1.7GHz, 9M L3)
- Microsoft Windows 2003 Enterprise (version RTM 3790)
- For spec2k INT suite, only 175.vpr and 252.eon executes significant numbers of SIMD instruction

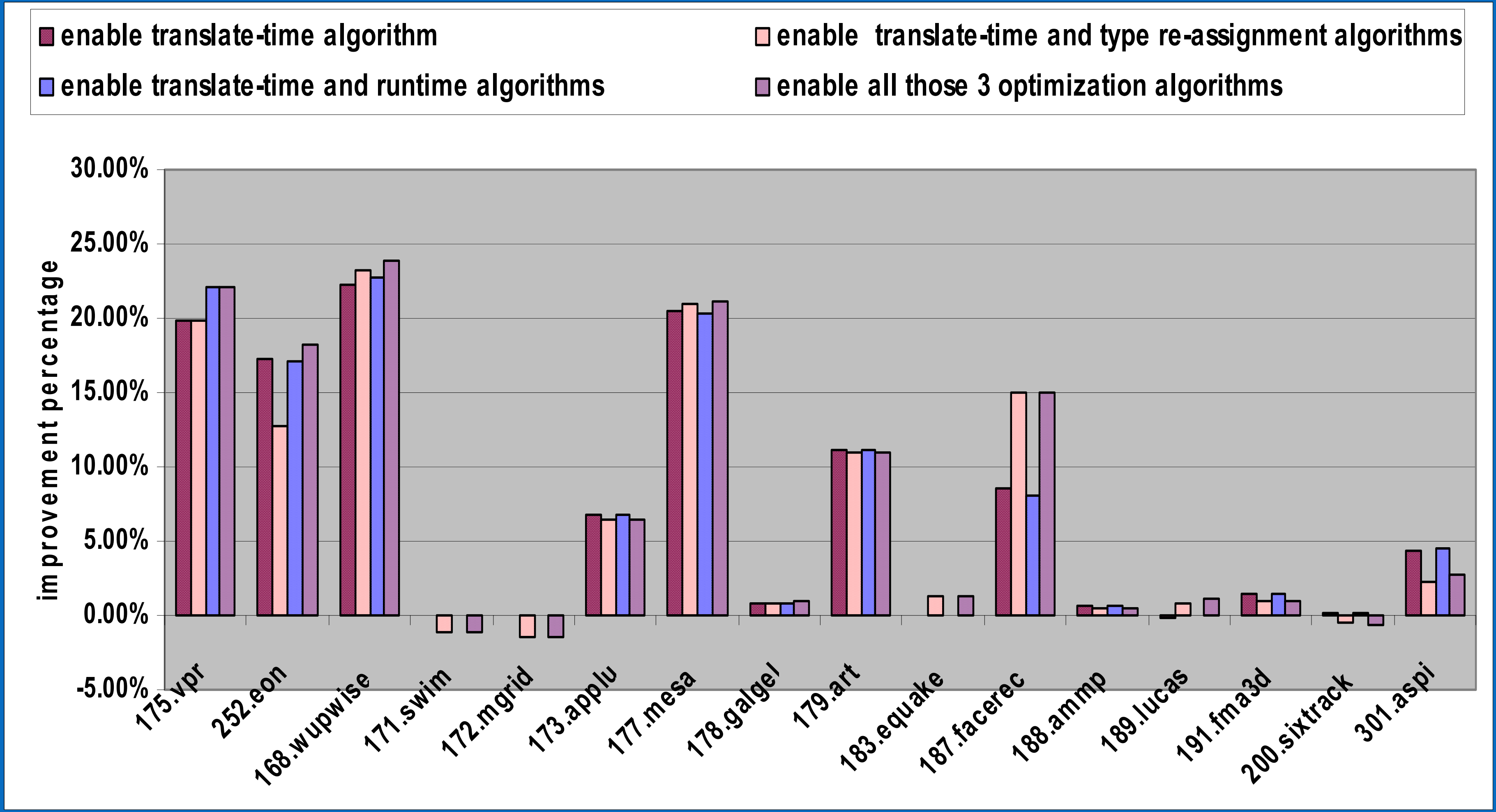
Effectiveness of the translate-time algorithm and the runtime algorithm

	Inter-block synchro-nization baseline (million)	Inter-block synchro-nizasion reduction ratio	exec time redu-ction (sec)	Speed-up ratio
175.vpr	1856.1	89.25%	55.9	22.1%
252.eon	867.6	99.99%	36.9	17.1%
168.wupwise	448.5	99.99%	59.1	22.8%
171.swim	0.9	98.69%	0.1	0.0%
172.mgrid	0.03	5.21%	0.0	0.0%
173.applu	269.6	100.00%	22.8	6.7%
177.mesa	290.0	99.99%	53.6	20.3%
178.galgel	7.7	95.73%	1.4	0.7%
179.art	541.1	99.97%	14.4	11.1%
183.equake	0.6	96.90%	0.0	0.0%
187.facerec	1408.7	99.99%	25.4	8.1%
188.ammp	47.8	99.88%	2.7	0.7%
189.lucas	13.2	99.96%	0.0	0.0%
191.fma3d	61.0	99.96%	5.0	1.4%
200.sixtrack	5.7	26.22%	0.3	0.1%
301.aspi	55.0	99.80%	18.6	4.4%

Effectiveness of the type re-assignment algorithm

	Reduced intra-block conversion count (million)	Transformed memory access count (million)	Actual speedup ratio
175.vpr	1.5	0.0	0.00%
252.eon	-545.3	16402.9	1.30%
168.wupwise	2213.3	7017.2	1.31%
171.swim	4281.0	4354.0	-1.22%
172.mgrid	-604.9	897.8	-1.44%
173.applu	-1782.1	11933.6	-0.33%
177.mesa	3247.6	1074.2	1.10%
178.galgel	40.4	177.3	0.16%
179.art	-0.1	975.7	-0.21%
183.equake	354.5	3091.9	1.27%
187.facerec	6398.0	1015.3	7.42%
188.ammp	-19.5	1250.7	-0.18%
189.lucas	-1686.3	3291.2	1.11%
191.fma3d	-880.6	16028.3	-0.50%
200.sixtrack	-2390.9	1592.5	-0.69%
301.aspi	-2324.7	10428.7	-1.71%

Overall speedup of SPEC2K



Conclusion and Acknowledgements

- This is the first paper describing how to support SIMD instructions on top of one product binary translator.
- A general baseline algorithm by performing both intra-block and inter-block data types synchronization.
- Three optimizations are implemented further to reduce the overhead of synchronization and to generate the lowest cost code.
- Thanks Cliff Young for providing insightful suggestions on revising this paper. And also thanks to those anonymous reviewers.
- Thanks to IA32-EL team, whose excellent work made our job possible

Q&A