Dalsoft
the tools to do the job

www.Dalsoft.com

There is a crack, a crack in everything
That's how the light gets in.
That's how the light gets in.
That's how the light gets in.

"Anthem"
Leonard Cohen

Exploring parallelism of modern processors
Auto-parallelizer for a multi core x86 architecture

created: July 2010

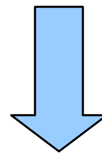
Exploring parallelism of modern processors

- **utilization of the instruction level parallelism**
  maximizing the use of execution units of a superscalar processor
- **use of SIMD**
  mapping the instruction stream to the SIMD instructions supported by the processor
  use of aligned memory access
- **auto-parallelizer for multi-core architecture**
  identifying code-patterns that are suitable for parallelization
  creating the optimized code that will be executed by all the cores available

dco is compiler post-processor

```
cc -c -O3 foo.c
```

```
.c.o:
    $(CC) $(CFLAGS) -c $<
```



```
cc -S -O3 foo.c
dco foo.s -o ofoo.s
mv ofoo.s foo.s
cc -c foo.s
rm foo.s
```

```
.c.o:
    $(CC) $(CFLAGS) -S $<
    dco $*.s -o $*.s $(DOPTS)
    $(CC) $(CFLAGS) -c $*.s
    rm $*.s
```

x86

AlphaPowered™

SHARC

STAR CORE
BRIGHTER DSP TECHNOLOGY!

- optimizes optimized code
- allows selective optimization
- is an optional tool

sources of optimization

instruction level parallelism
multi-core architecture

foundation of the optimizer

resource constrain sets
abstract resource representation
code splitting

instruction level parallelism

super-scalar execution using multiple execution units

sequential nature of the code is preserved

Dalsoft
the tools to do the job

www.Dalsoft.com

Compare **dco** with the Intel's **icc** compiler

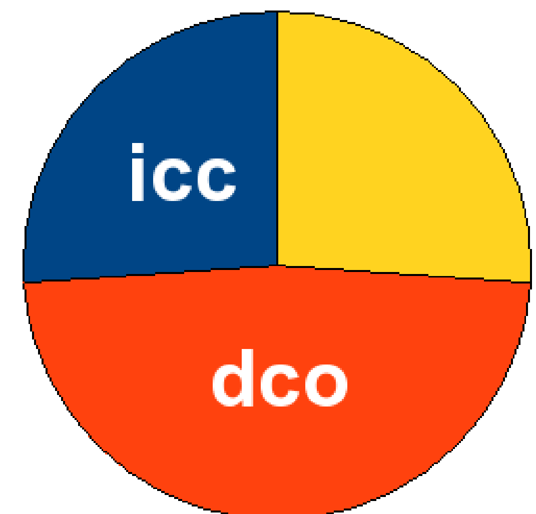
Kernel#	gcc	gcc+dco	icc	gcc+dco/gcc	icc/gcc	icc/gcc+dco
1	4.97	3	2.96	39.64%	40.44%	1.33%
2	2.38	2.34	2.28	1.68%	4.20%	2.56%
3	5.93	2.33	2.54	60.71%	57.17%	-9.01%
4	4.66	3.79	4.63	18.67%	0.64%	-22.16%
5	5.2	1.75	2.38	66.35%	54.23%	-36.00%
6	4.53	3.55	3.87	21.63%	14.57%	-9.01%
7	4.87	3.12	2.57	35.93%	47.23%	17.63%
8	5	3.87	3.25	22.60%	35.00%	16.02%
9	4.6	3.86	4.97	16.09%	-8.04%	-28.76%
10	4.94	3.38	4.32	31.58%	12.55%	-27.81%
11	5.78	0.93	1.65	83.91%	71.45%	-77.42%
12	5.18	4.42	4.13	14.67%	20.27%	6.56%
13	4.57	4.62	4.61	-1.09%	-0.88%	0.22%
14	4.71	4.12	2.3	12.53%	51.17%	44.17%
15	3.72	3.73	3.67	-0.27%	1.34%	1.61%
16	5.61	5.32	5.66	5.17%	-0.89%	-6.39%
17	5.01	4.98	4.86	0.60%	2.99%	2.41%
18	4.7	3.74	3.45	20.43%	26.60%	7.75%
19	5.81	4.1	6.77	29.43%	-16.52%	-65.12%
20	4.53	4.43	4.38	2.21%	3.31%	1.13%
21	4.88	4.6	1.05	5.74%	78.48%	77.17%
23	4.17	3.85	4.67	7.67%	-11.99%	-21.30%
24	4.85	0.78	1.66	83.92%	65.77%	-112.82%
GM	4.74	3.21	3.29	32.33%	30.56%	-2.61%

Livermore loops benchmark

The best results are shown in this color

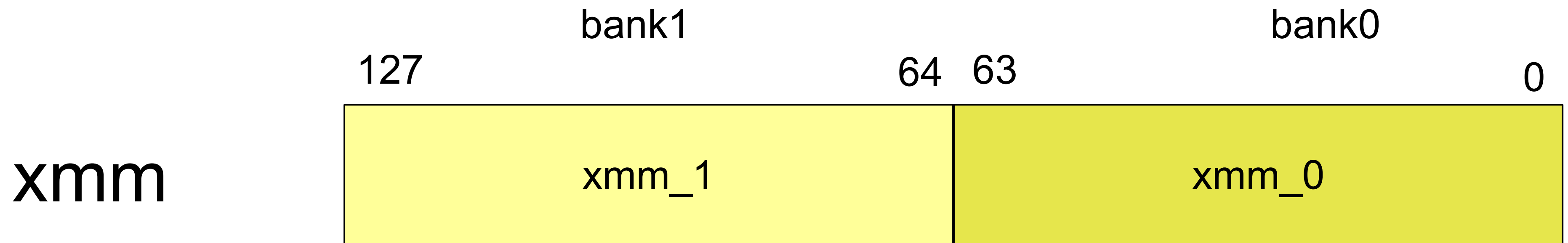
count of the number of the best results:

icc 6
dco 11



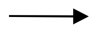
x86 offers the 'Single Instruction Multiple Data' (SIMD) instructions. The part of dco utilizing these instructions is called

SIMDinator (pronounced *seem-d-ney-ter*)

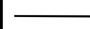


mulsd %xmm5,%xmm6	xmm6_0 = xmm6_0*xmm5_0
mulpd %xmm5,%xmm6	xmm6_0 = xmm6_0*xmm5_0 xmm6_1 = xmm6_1*xmm5_1
mul_d %xmm5_z,%xmm6_y	xmm6_y = xmm6_y*xmm5_z

.
mulsd %xmm2,%xmm3
.
mulsd %xmm4,%xmm5
.



.
mul_d %xmm2_0,%xmm3_0
.
mul_d %xmm4_0,%xmm5_0
.



.
mul_d %xmm6_1,%xmm2_0
mul_d %xmm3_0,%xmm4_1
.
.

resource constrain equations

BasicConstrainSet (BCS)

SIMD ConstrainSet (CS)

mul_d %xmm6_1,%xmm2_0: i_1 o_{11}, o_{12}

mul_d %xmm3_0,%xmm4_1: i_2 o_{21}, o_{22}

SameBank(o_{11}, o_{12})
SameBank(o_{21}, o_{22})
SameRegister(o_{11}, o_{21})
SameRegister(o_{12}, o_{22})

SameBank(xmm5_1,xmm3_1)=true
SameBank(xmm6_0,xmm4_0)=true
SameBank(xmm7_1,xmm2_0)=false

SameRegister(xmm5_1,xmm5_0)=true
SameRegister(xmm6_0,xmm6_0)=true
SameRegister(xmm7_1,xmm4_1)=false

.
mul_d %xmm7_1,%xmm3_1
mul_d %xmm7_0,%xmm3_0
.

.
mulpd %xmm7,%xmm3
.
.

Dalsoft
the tools to do the job

www.Dalsoft.com

Kernel#	gcc	gcc+dco	-np	gcc+dco/gcc	-np/gcc	-np/gcc+dco
1	4.96	3.32	4	33.06%	19.35%	-20.48%
2	2.38	2.32	2.32	2.52%	2.52%	0.00%
3	5.93	3.55	3.84	40.13%	35.24%	-8.17%
4	4.66	4.12	3.8	11.59%	18.45%	7.77%
5	5.2	2.07	2.07	60.19%	60.19%	0.00%
6	4.53	3.9	3.63	13.91%	19.87%	6.92%
7	4.87	3.12	3.96	35.93%	18.69%	-26.92%
8	5	5.4	3.88	-8.00%	22.40%	28.15%
9	4.6	3.95	4.23	14.13%	8.04%	-7.09%
10	4.94	3.87	3.38	21.66%	31.58%	12.66%
11	5.78	1.52	1.52	73.70%	73.70%	0.00%
12	5.18	4.98	4.39	3.86%	15.25%	11.85%
13	4.57	5.56	4.58	-21.66%	-0.22%	17.63%
14	4.71	4.71	4.26	0.00%	9.55%	9.55%
15	3.72	3.72	3.72	0.00%	0.00%	0.00%
16	5.61	5.31	5.29	5.35%	5.70%	0.38%
17	5.01	4.99	4.99	0.40%	0.40%	0.00%
18	4.7	3.96	3.95	15.74%	15.96%	0.25%
19	5.81	4.1	4.1	29.43%	29.43%	0.00%
20	4.53	4.43	4.43	2.21%	2.21%	0.00%
21	4.88	4.61	4.61	5.53%	5.53%	0.00%
22	4.88	6.21	4.86	-27.25%	0.41%	21.74%
23	4.17	4.09	4.09	1.92%	1.92%	0.00%
24	4.85	0.77	0.77	84.12%	84.12%	0.00%
Geometric Mean	**4.75**	**3.64**	**3.53**	23.37%	25.68%	3.02%

makes code faster

makes code slower

prevents faster code generation

auto-parallelizer for multi-core architecture

suitable for shared memory multi-core systems with uniform memory access (UMA)
multiple identical processors (cores) on a single chip
that have equal access and access times to memory

utilizes threads model with a single process creating and managing
multiple threads that are executed in parallel by different cores
implemented by calling subroutines from the OpenMP library

abstract resource representation

$rsrc=c_0+\sum c_n*rsrc_n$

rsrc – register or memory reference

- set of resources depends on the code that is optimized
- each resource is a “set” of values
- resources are kept as vectors

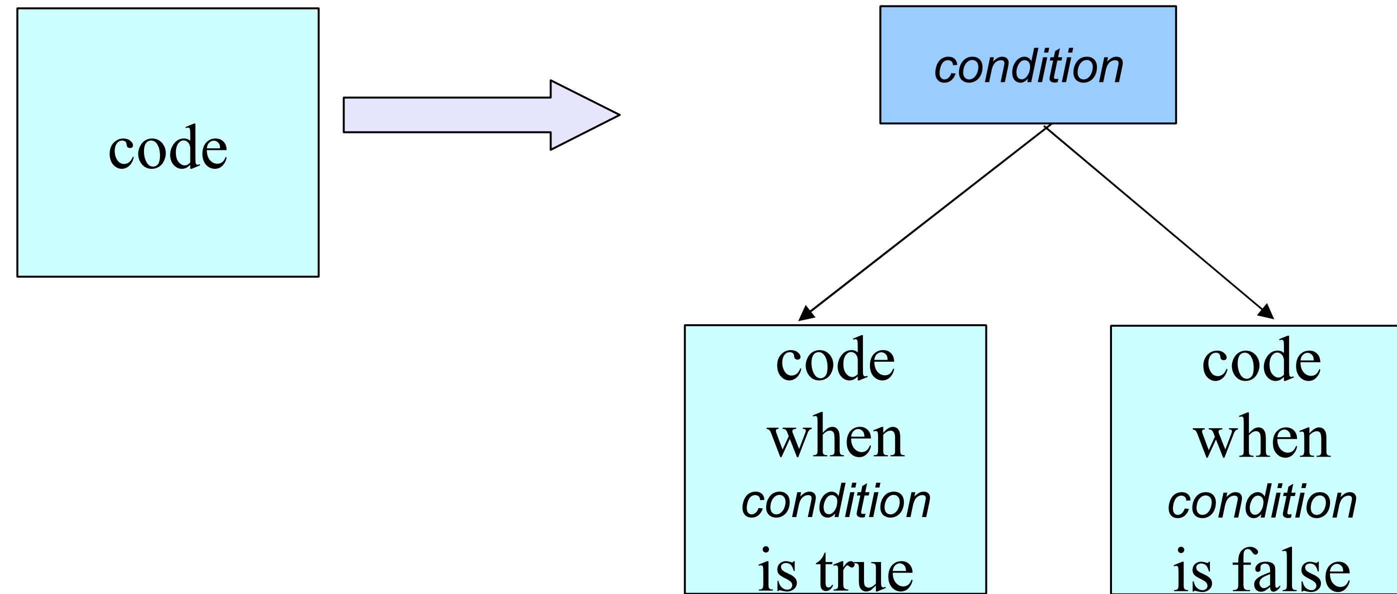
.
r8=r9+r10
.

rsrc – register or memory reference

conditional code execution

static

dynamic

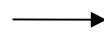


depending on the way *condition* is calculated
- dynamic static
- dynamic dynamic

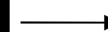
used in
- align memory generation
- cost analysis
- memory disambiguation (anti-aliasing)

code splitting

```
for ( i = 0; i < ARRAY_DIM; i++ ) {
 c[i] = exp( wrap );
 wrap += a[i];
}
```



```
for ( i = i_n; i < i_n+1; i++ ) {
 c[i] = exp( wrap );
 wrap += a[i];
}
```



```
for ( i = 0; i < i_n; i++ ) {
 wrap += a[i];
}
for ( i = i_n; i < i_n+1; i++ ) {
 c[i] = exp( wrap );
 wrap += a[i];
}
```

applied to

- set of registers
- memory regions

used in

- set up for a thread execution
- dynamic dynamic memory disambiguation

results of auto-parallelization

- parallelizes code that can not be processed by OpenMP
- parallelizes great number of kernels (dot-product etc.)
- parallelzes applications (tomcatv, alvinn etc.)

cost of auto-parallelization

	gcc	**dco**	**omp**	dco/omp
alvinn	0.428	0.361	0.403	10.42%
Poisson equation solver (*dynamic dynamic* disambiguation)	58	56	52	-7.69%

optimization is obsolete!

original time	**optimized time**	**parallel time (10 cores)**	**optimized parallel time**	optimized	parallel	optimized parallel	difference
100	80	10	8	20.00%	90.00%	92.00%	2.00%

the addition of the the serial code improvement **α** on a **n** core system is

α/n

or is it really obsolete?

- not all code can be parallelized
- some code has to be modified before it can be parallelized
- develop thread-optimization technology

Dalsoft
the tools to do the job

Thank you!