Performance of the Dot Product Function in Radiative Transfer Code SORD

Sergey Korkin (USRA GESTAR)*

Alexei Lyapustin (NASA GSFC)

Aliaksandr Sinyuk (Sigma Space Corp.)

Brent Holben (NASA GSFC)

*sergey.v.korkin@nasa.gov

$$\mathbf{A}\cdot\mathbf{B} = \sum_{i=1}^{N} A_i B_i$$

Disclaimer

- I am not an IT specialist… My background is electro-optical systems and atmospheric optics. But currently I am facing a problem of scientific software performance enhancement.

- So, please feel free to advise better compiler keys, freeware libraries, tricks, …

- **No OpenMP** is considered in this presentation. With parallel computing one would get **similar conclusion, but faster** (I think so).

Radiative Transfer (RT) Code

- Numerically simulates scattering of light in planetary atmospheres, ocean, etc.
- Used in retrieval algorithms – scientific software that fits measurements and numerical simulations by adjusting input for the RT code, and thus retrieves parameters of scattering media: atmospheric aerosol, clouds, etc.
- Must be efficient: accurate (*enough*) and **fast** (*invoked hundreds, thousands, … times*)

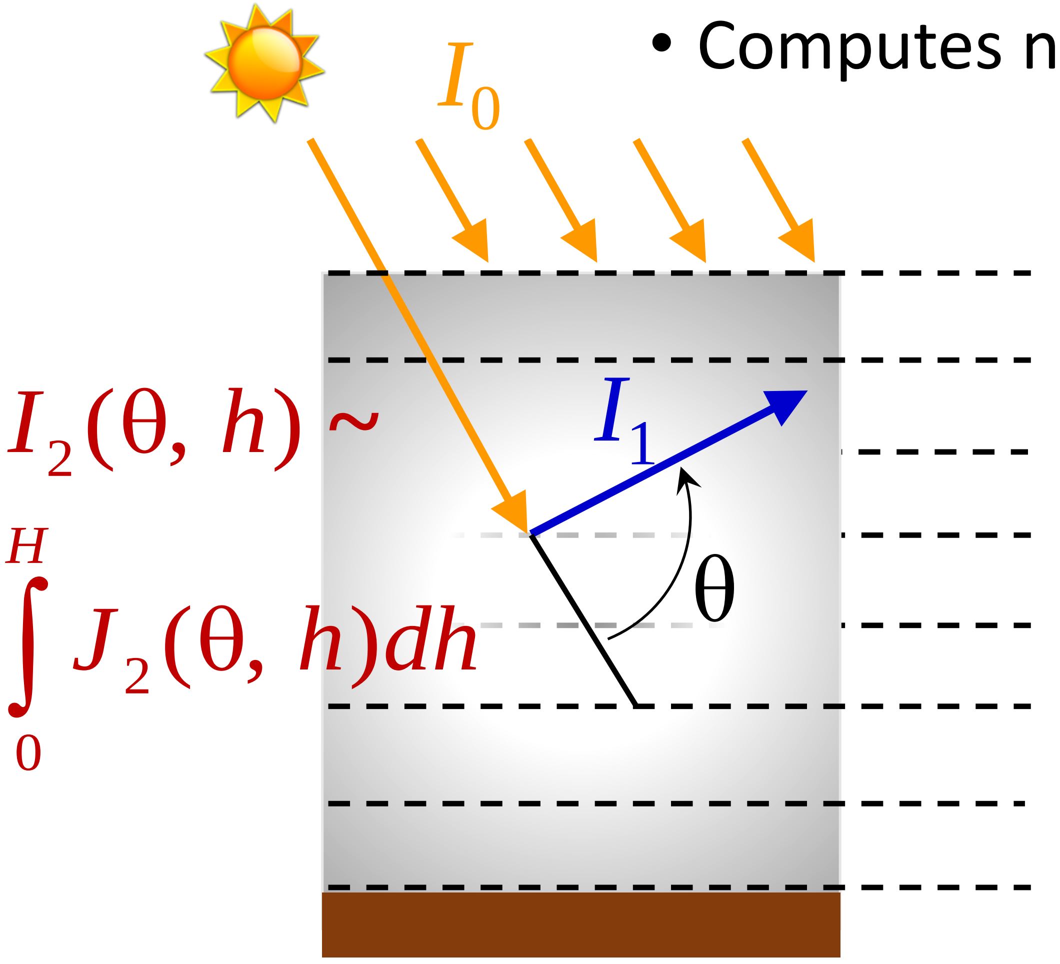
RT Code SORD (SPIE,v9853,2016)

- Includes many features of realistic atmosphere: height profiles, surface reflection, polarization of light, etc;
- Used by the NASA GSFC AERONET team;
- Tested against 50 published benchmarks using ifort, pgf90, *gfortran*;
- Publicly available from ftp://maiac.gsfc.nasa.gov/pub/skorkin/
  or by email request from sergey.v.korkin@nasa.gov;
- Uses the known method of **S**ucessive **ORD**eres of scattering

Successive Orders (SO)

- Computes next order from the previous one numerically;

$$I(\theta,h) = I_1(\theta,h) + I_2(\theta,h) + I_3(\theta,h) + \ldots$$



$$J_2(h_i) = \int p(\theta,h_i) I_1(\theta,h_i) d\theta$$

- Relatively simple for coding;
- Developed and widely used;
- Does not require external libs;
- Has clear physical background.

Dot Product in the SO

- Scattering at each level and in each direction – Gauss summation

$$\int_{-1}^{1} p(\mu_i,\mu) I(\mu) d\mu \approx \sum_{j=1}^{N} w_j p(\mu_i,\mu_j) I(\mu_j) = \mathbf{P}\cdot\mathbf{I} = dot\_product(\mathbf{P},\mathbf{I})$$

- Estimation of number of the dot product calls:

  100 Levels x 50 View directions x 10 Azimuth (Fourier) moments x 10 Scattering orders x 9 elements of the 3-by-3 Mueller matrix (polarization) = 4.5M calls per wavelength per single run

- Spectral measurements & derivatives – efficient dot product needed

Implementation

- Direct (*is it a good idea to allow compiler to unroll loops ?)*

```
DOT = 0.0
DO IX = 1, N
      DOT = S + &
      X1(IX)*X2(IX)
END DO
```

- To reduce loop overhead (change/check index, `IX`) use >>

- >> Unrolled loops – factor 3

```
DOT = 0.0
M = MOD(NX, 3)
DO IX = 1, M
      DOT3 = DOT3 +
      X1(IX)*X2(IX)
END DO
M1 = MX+1
DO IX = M1, NX, 3
      DOT3 = DOT3 +          &
      X1(IX)*X2(IX) +        &
      X1(IX+1)*X2(IX+1) +    &
      X1(IX+2)*X2(IX+2)
END DO
```

Expert Opinion: _DOT from BLAS

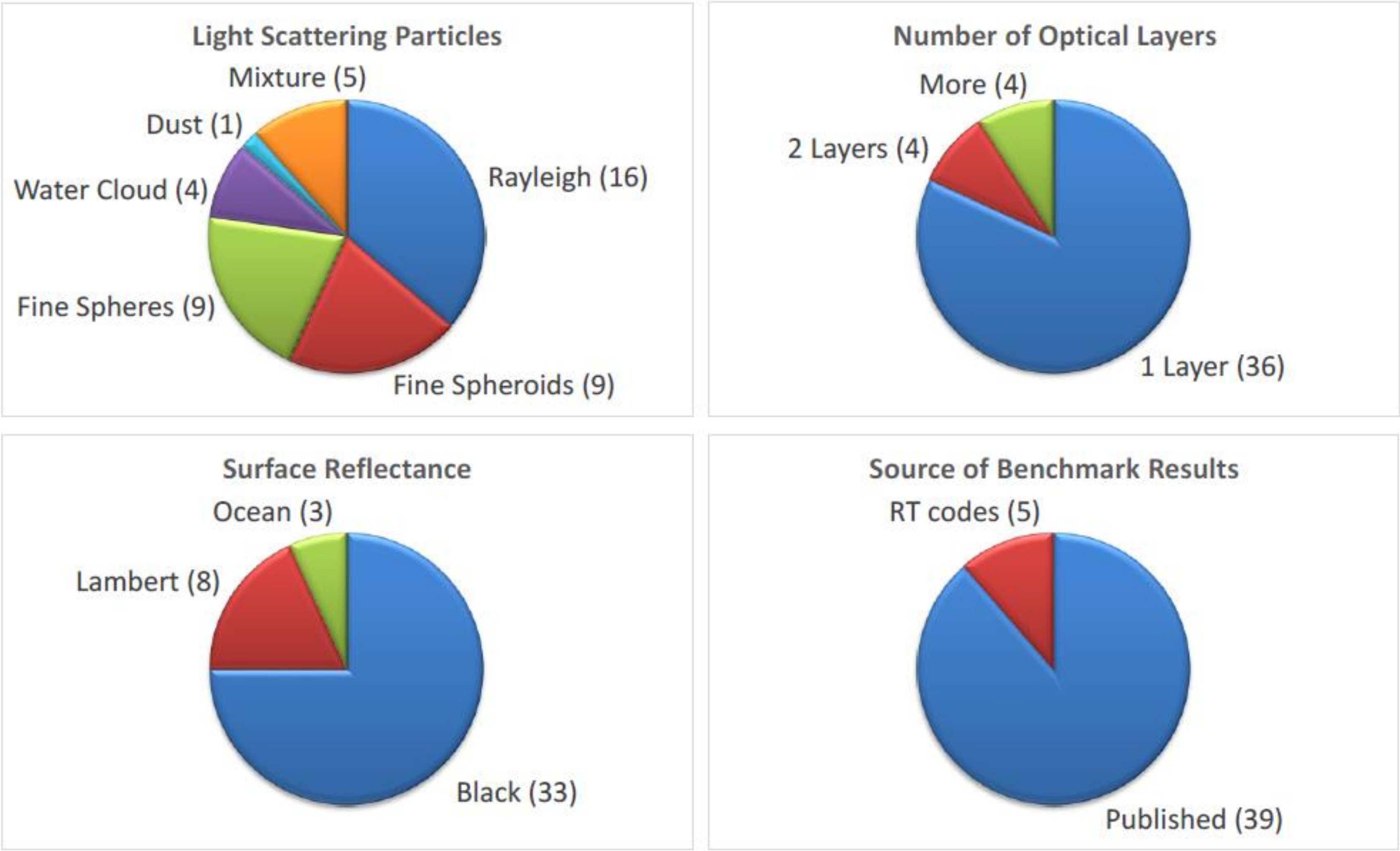
```
*          clean-up loop
*
           m = mod(n,5)
           IF (m.NE.0) THEN
              DO i = 1,m
                 dtemp = dtemp + dx(i)*dy(i)
              END DO
              IF (n.LT.5) THEN
                 ddot=dtemp
              RETURN
              END IF
           END IF
           mp1 = m + 1
           DO i = mp1,n,5
            dtemp = dtemp + dx(i)*dy(i) + dx(i+1)*dy(i+1) +
     $              dx(i+2)*dy(i+2) + dx(i+3)*dy(i+3) + dx(i+4)*dy(i+4)
           END DO
```

1. Is the factor 5 *always* the best ?

2. If not, which one is the best ?

3. Why 5 … ? I don't know…

Benchmark Scenarios



- Korkin et al. (2016), SPIE v.9853, 985305 reports 44 benchmarks;
- + 6 new benchmarks including realistic height profiles: see Korkin et al., This Conference, Paper No. 10001-10;
- **50 scenarios total**.

Implementations of DDOT

- Direct implementation: $A_1*B_1+A_2*B_2 + \ldots + A_N*B_N$
- Unrolled loops with a factor of 2, 4, 8, 16 (Gauss quadrature)
- Built in Fortran **DOT_PRODUCT(A, B)** and **SUM(A*B)**
- BLAS DDOT: unrolling factor 5
- BLAS DDOT for both increments = 1: DDOT(N, DX, INCX, DY, INCY)

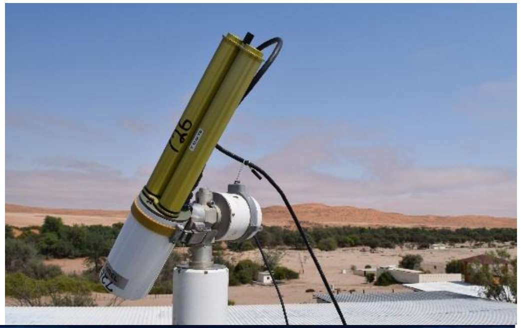
See e.g. Severence & Dowd, 1998; Hager & Wellein, 2011 etc.

Hardware & Software

Machine 1 = "*ifort*": Intel® i7-2720QM CPU, 2.2GHz, Windows 7 64 bit; **Intel Visual Fortran Compiler** 11.0.072 integrated with Microsoft Visual Studio 2008. Configure Optimization for "Maximize Speed". The RT code SORD was developed on Machine 1.

Machine 2 = "*pgf 90*": Intel® Xeon E7-4890 v2 CPU, 2.8 GHz, Linux 2.6 64 bit; The **Portland Group Fortran 90/95 compiler** 7.1-4.
Compiler keys: *-O3 –Mipa=fast, inline = Msmartalloc*.

The NASA GSFC AERONET team uses this machine for data processing and research.



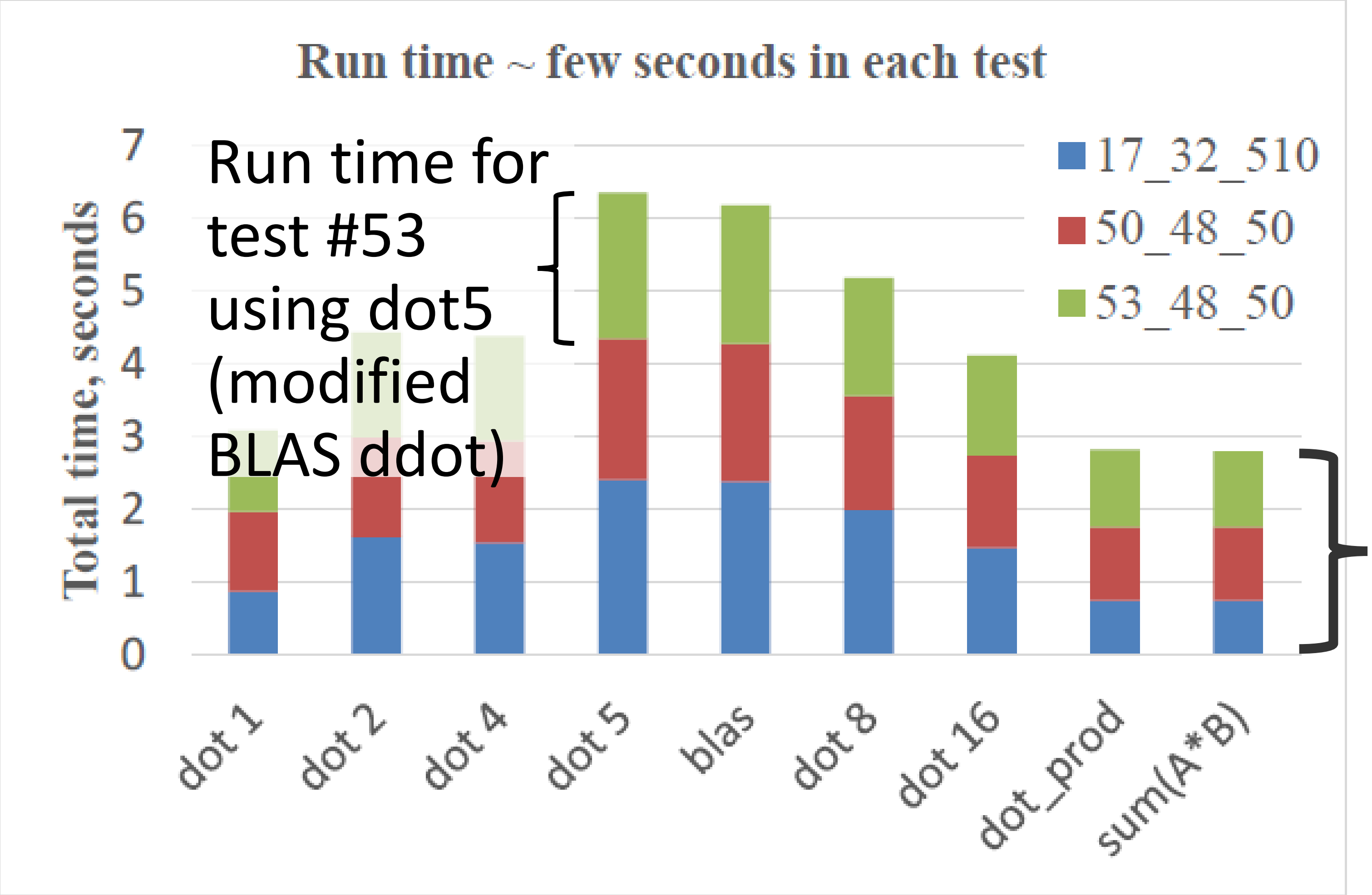
Understanding of Results

Timing:

On both machines - CPU_TIME from Fortran;

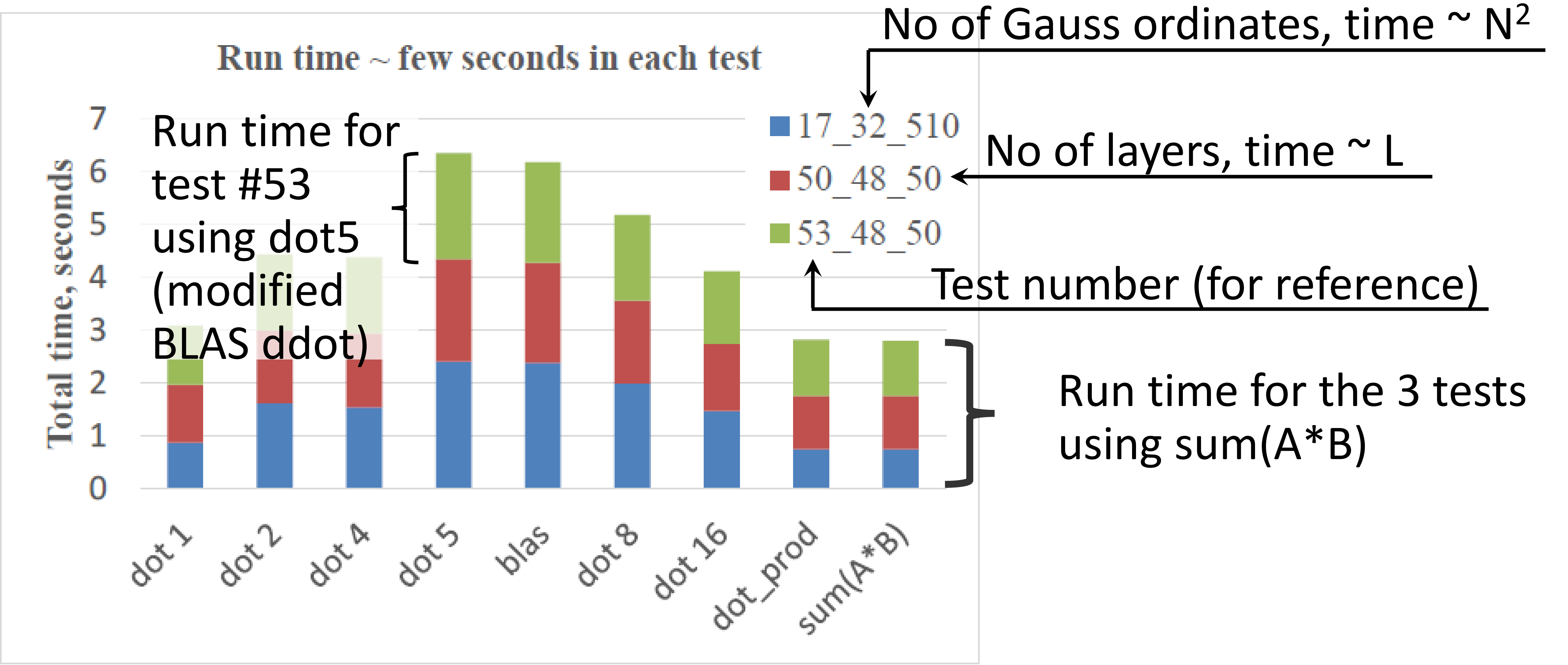
On the Linux machine – *time* command (close to the CPU_TIME readings)

Understanding of Results

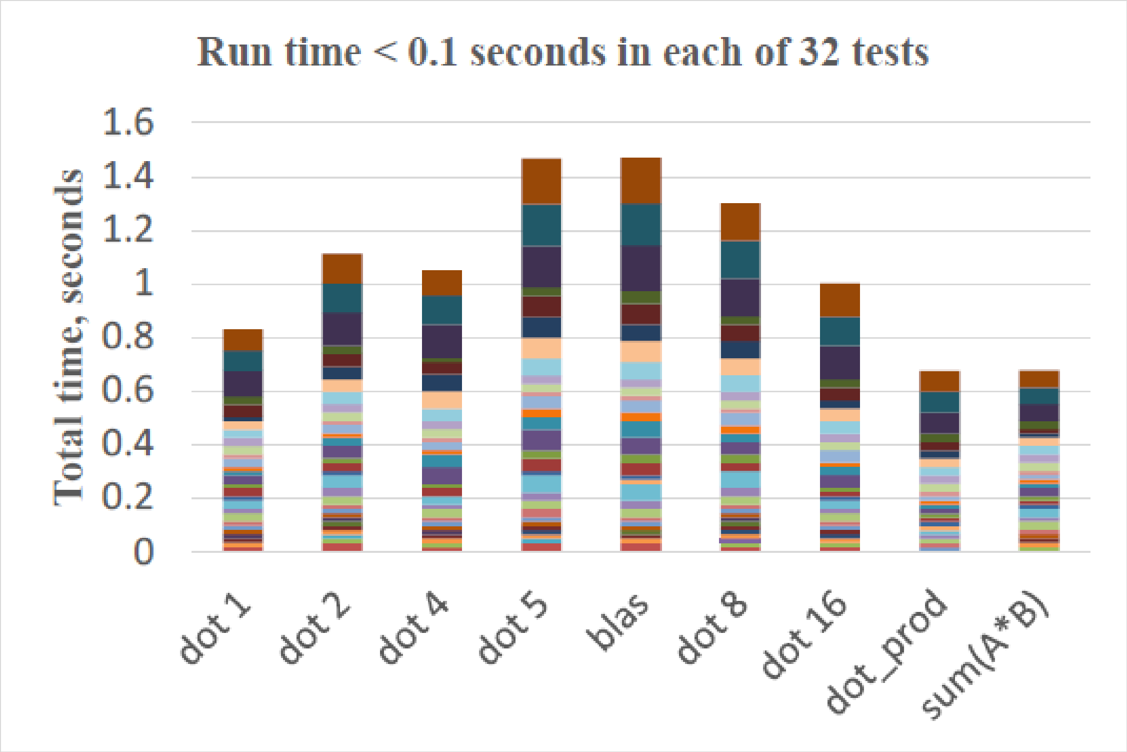


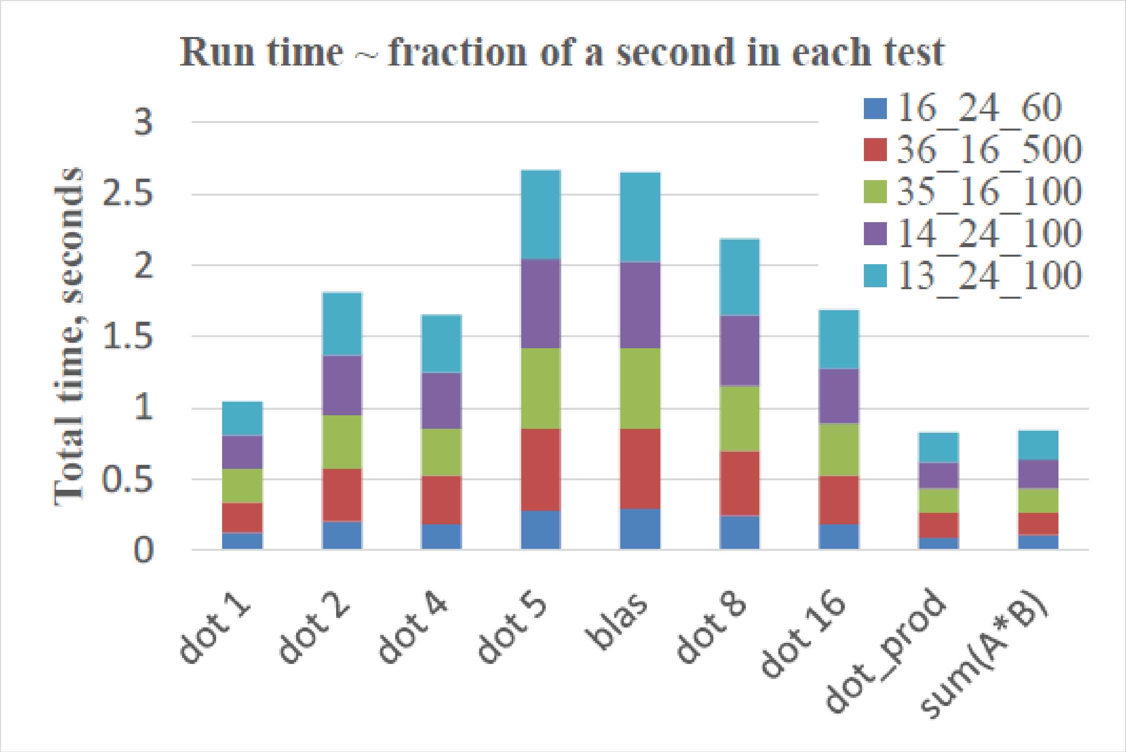
Run time for the 3 tests using sum(A*B)

Understanding of Results

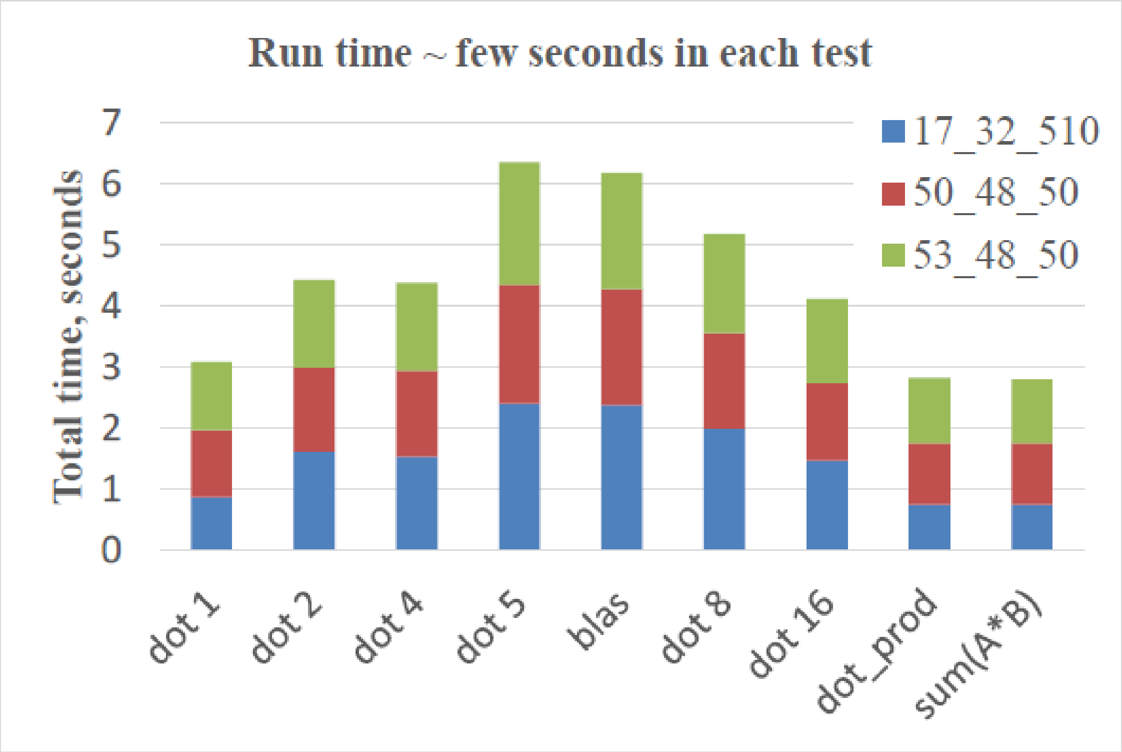


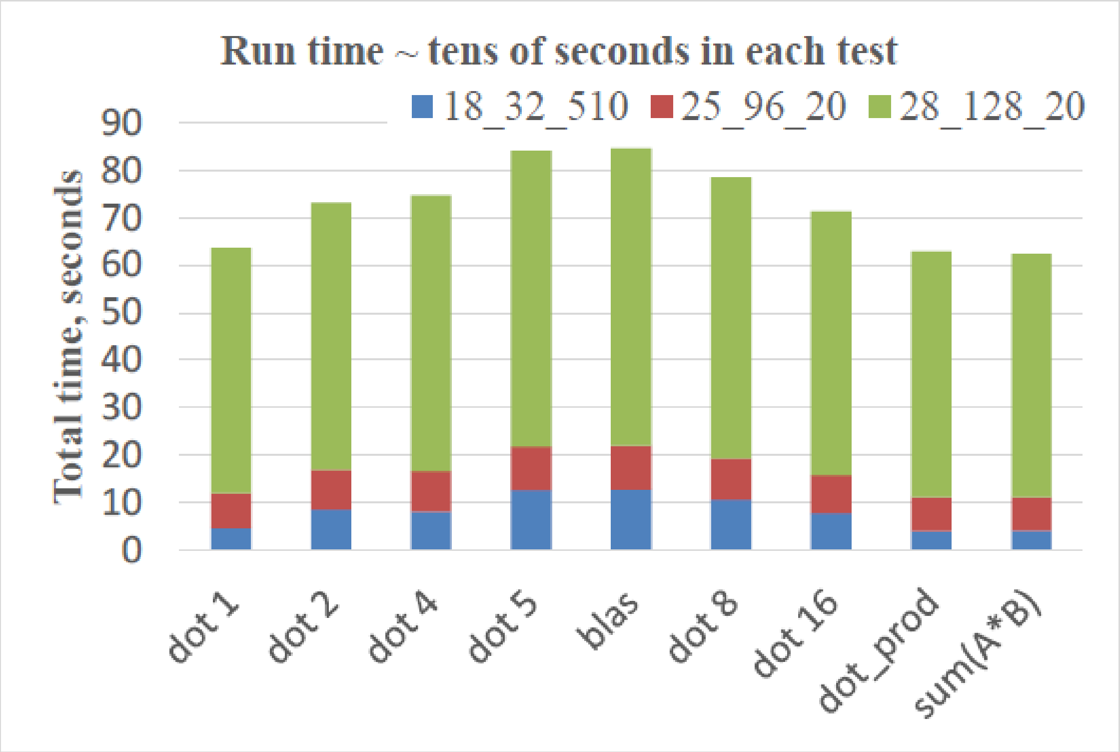
ifort: slide 1



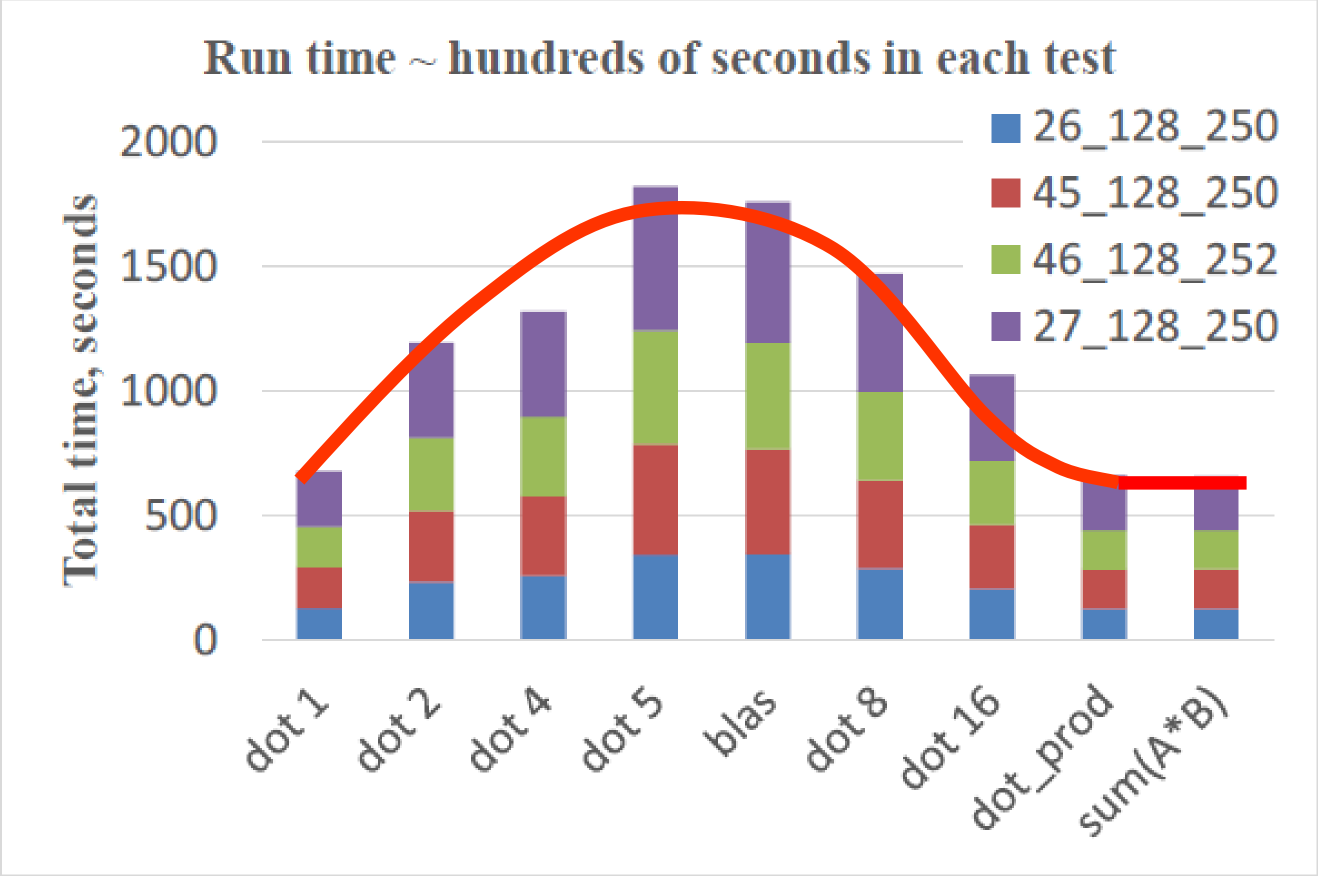


ifort: slide 2



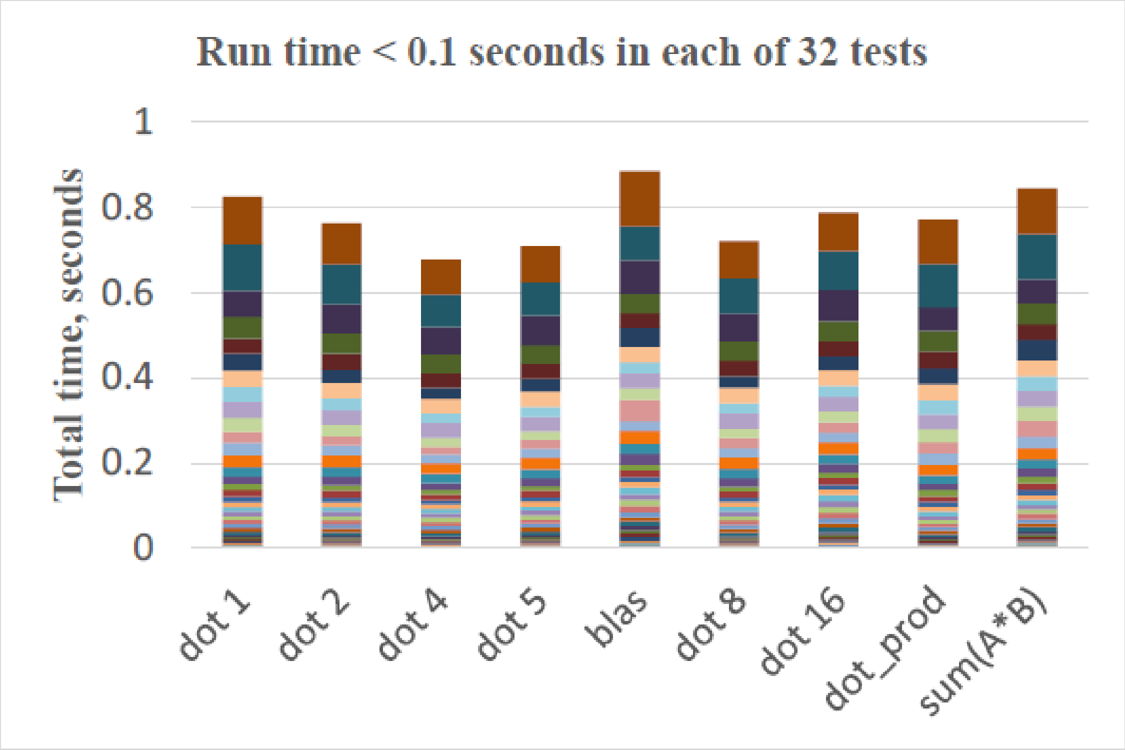


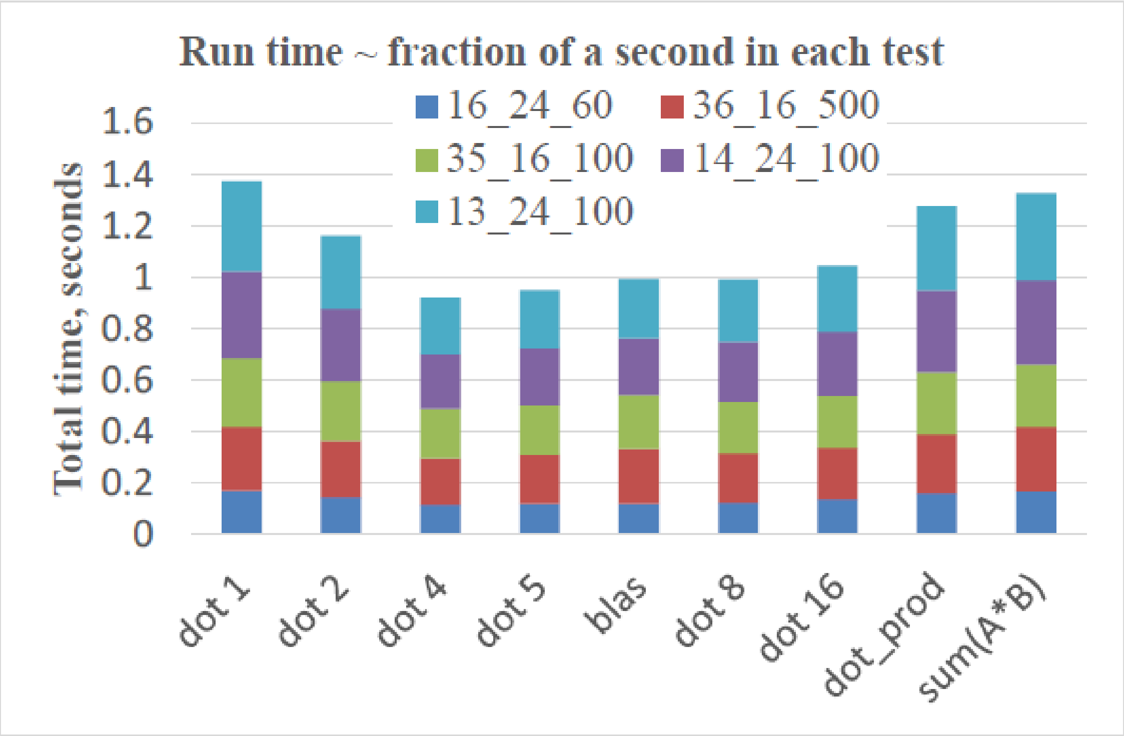
ifort: slide 3



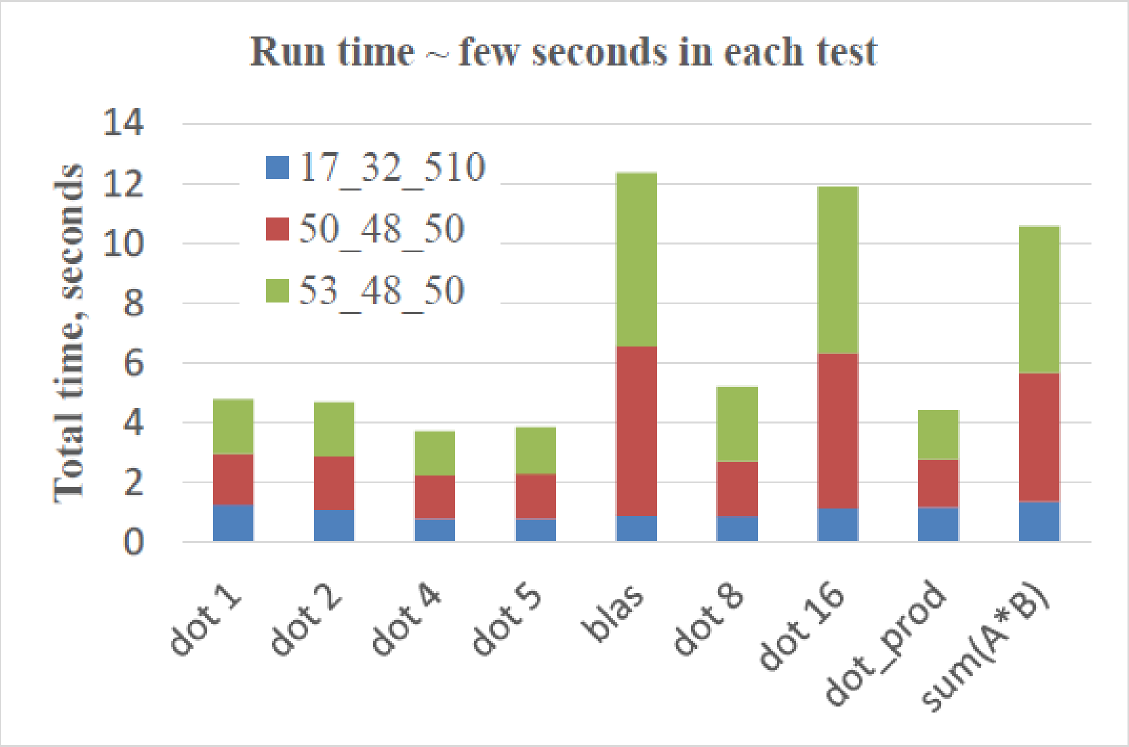
- On the ifort machine (Intel CPU + Intel Fortran compiler), the built-in Fortran dot product function shows the best performance;
- Unrolled loops, dot1, shows comparable performance;
- The BLAS ddot and dot5 show the worst performance in all test scenarios.

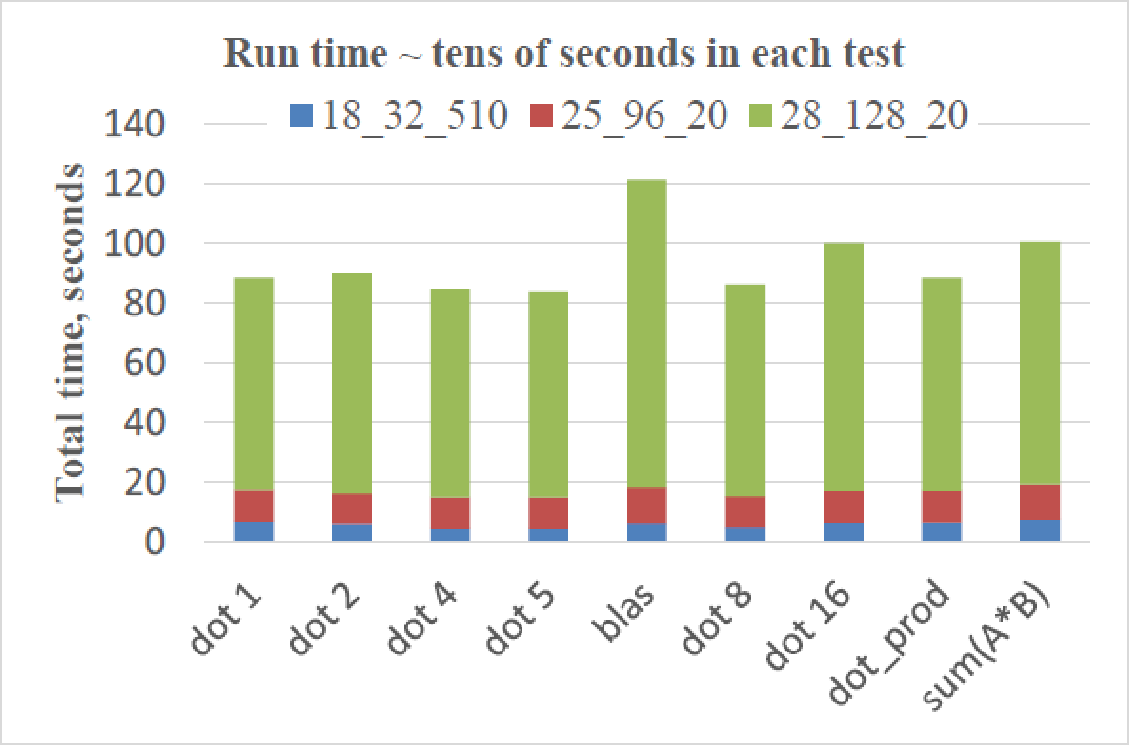
Pgf90 – slide 1



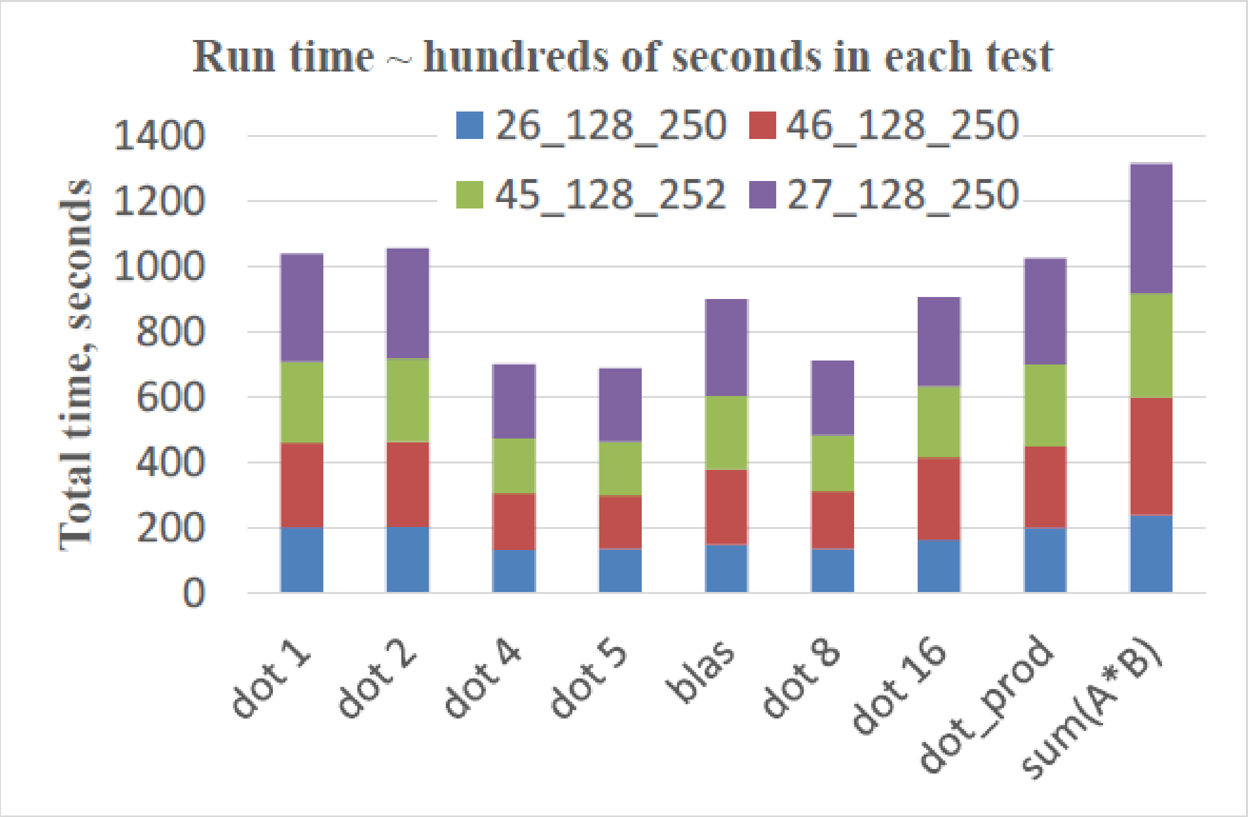


Pgf90 – slide 2





Pgf90 – slide 3



- On the pgf90 machine, the BLAS ddot is the least efficient;
- The built-in functions are not efficient either;
- ddot4 shows the best overall performance; ddot5 (simplified BLAS ddot) performs similar to ddot4.

“Food” for Thoughts

- DDOT from BLAS seems to be inefficient (created 1978, modified 1993). What about other subroutines: **M*M**, **1/M**, SVD frequently used in RT codes?

- Optimization of BLAS/LAPACK is time consuming and soft- & hardware dependent. Using of commercial Intel MKL, NAG limits the open-source distribution of RT codes. ATLAS? Any other open-source libraries?

+1 Way for Better Performance

- Parallel computation of the dot product (precondition loop is omitted). The four SUMs are independent. To be tested with RT code SORD soon…

```
SUM1 = 0.0
SUM2 = 0.0
SUM3 = 0.0
SUM4 = 0.0
DO IX = 1, NX, 4
      SUM1 = SUM1 + X1(I)  *X2(I)
      SUM2 = SUM2 + X1(I+1)*X2(I+1)
      SUM3 = SUM3 + X1(I+2)*X2(I+2)
      SUM4 = SUM4 + X1(I+3)*X2(I+3)
END DO
DOT = SUM1 + SUM2 + SUM3 + SUM4
```

*Dowd K., **1993**: High Performance Computing, O'Reilly & Assoc. Inc., p.203*

Gerber R, et al: 2006: The Software Optimization Cookbook, Intel Press, p.150

Conclusion

- Ifort's DOT_PRODUCT showed the best performance (not surprising);
- Performances of the BLAS DDOT is disappointing on both machines (what about the whole BLAS/LAPACK? Any tests published?)
- Dot product with unrolling factor 4, **DOT4**, seems to be the best for RT simulations using RT code SORD under **Linux+pgf90**;
- Optimization must be done in a wide range of scenarios. The new open-source RT code SORD comes with a package that allows for testing in a wide range of scenarios: ftp://maiac.gsfc.nasa.gov/pub/skorkin/

Acknowledgements

- This research is supported by the NASA ROSES-14 program “Remote Sensing Theory for Earth Science” managed by Dr. Lucia Tsaoussi, grant number NNX15AQ23G.

- Sergey Korkin thanks James Limbacher (SSAI and NASA GSFC, USA) and Dmitry Efremenko (DLR, Germany) for fruitful discussions on HPC.

Please send your critical feedback to sergey.v.korkin@nasa.gov

Thank you all for attention!