technology
from seed

Transparent application acceleration by intelligent scheduling of shared library calls on heterogeneous systems

João Colaço, **Adrian Matoga**, Aleksandar Ilic, Nuno Roma, Pedro Tomás, Ricardo Chaves

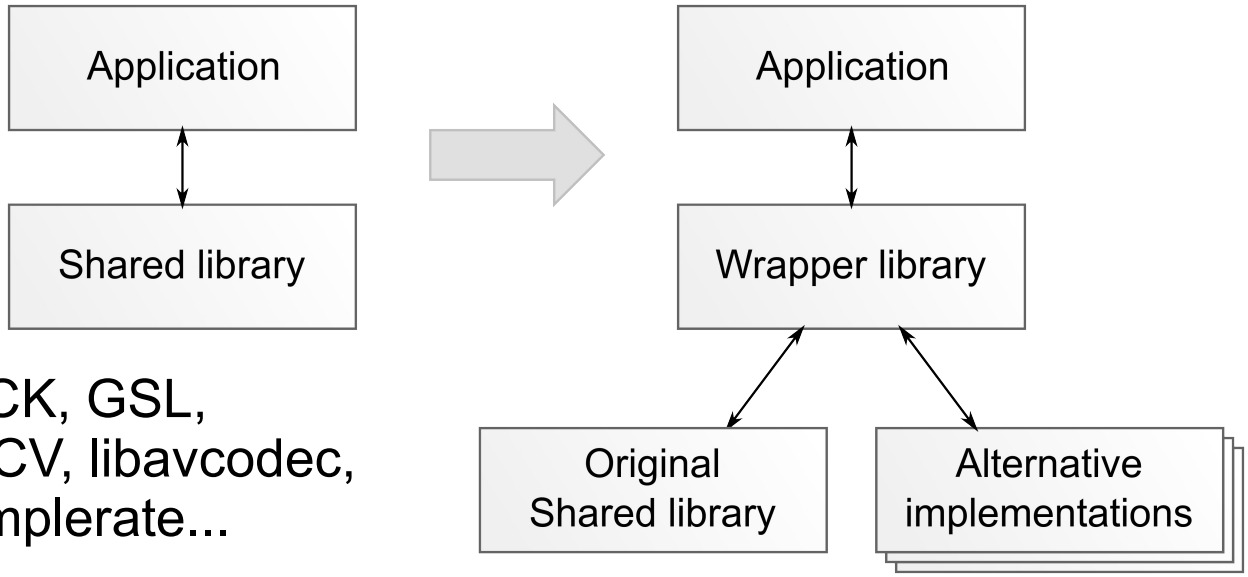
adrian.matoga@inesc-id.pt

Signal Processing Systems
INESC-ID / IST Portugal

PPAM 2013 – 10th International Conference on Parallel Processing and Applied Mathematics
September 8 – 11, 2013, Warsaw, Poland

Motivation

- **Transparent** application acceleration by *shared library interposing*



BLAS, LAPACK, GSL, FFTW, OpenCV, libavcodec, libjpeg, libsamplerate...

GPU, FPGA...

First proposed by Beisel et al. 2010

Improvements

- Select the optimal implementation using a **dynamic performance model**.
- Where possible, **balance the load** across multiple processors.

Outline

- **Framework Architecture**
  - Selection and Partitioning policies
  - Library Generation
- **Run-time adaptive policies**
  - Best-performance selection
  - Load balancing
- **Experimental results**
  - Accelerating BLAS and FFTW
  - Overheads

Outline

- **Framework Architecture**
  - Selection and Partitioning policies
  - Library Generation

Run-time adaptive policies

Best-performance selection

Load balancing

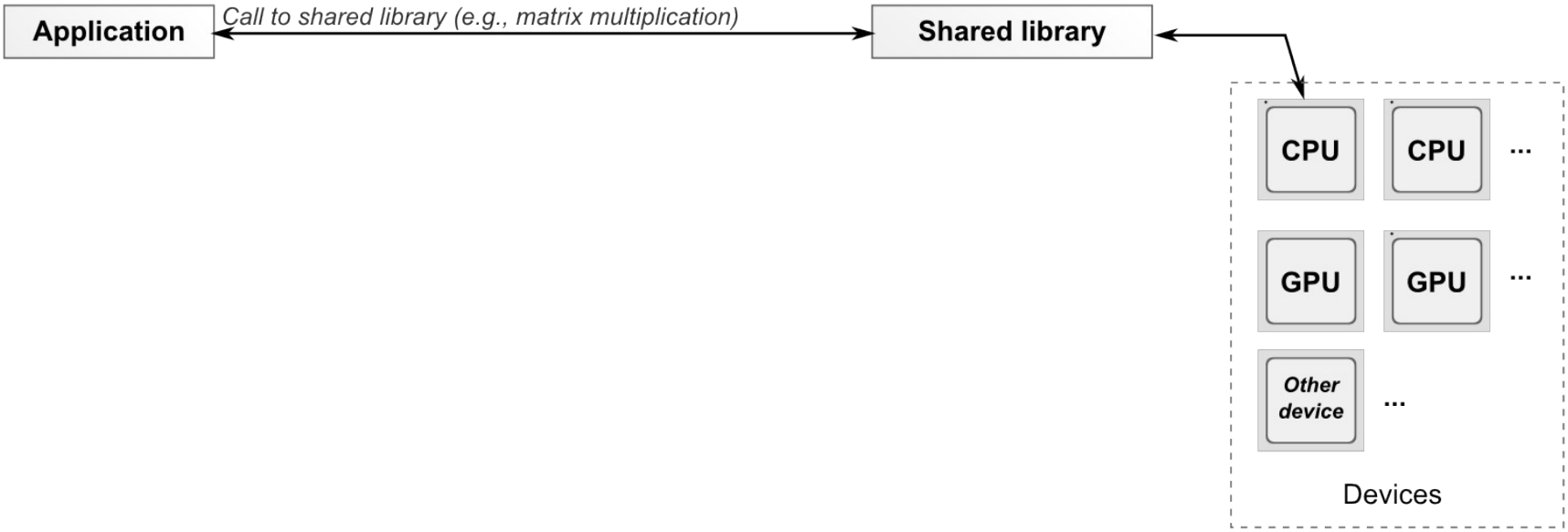
Experimental results

Accelerating BLAS and FFTW

Overheads

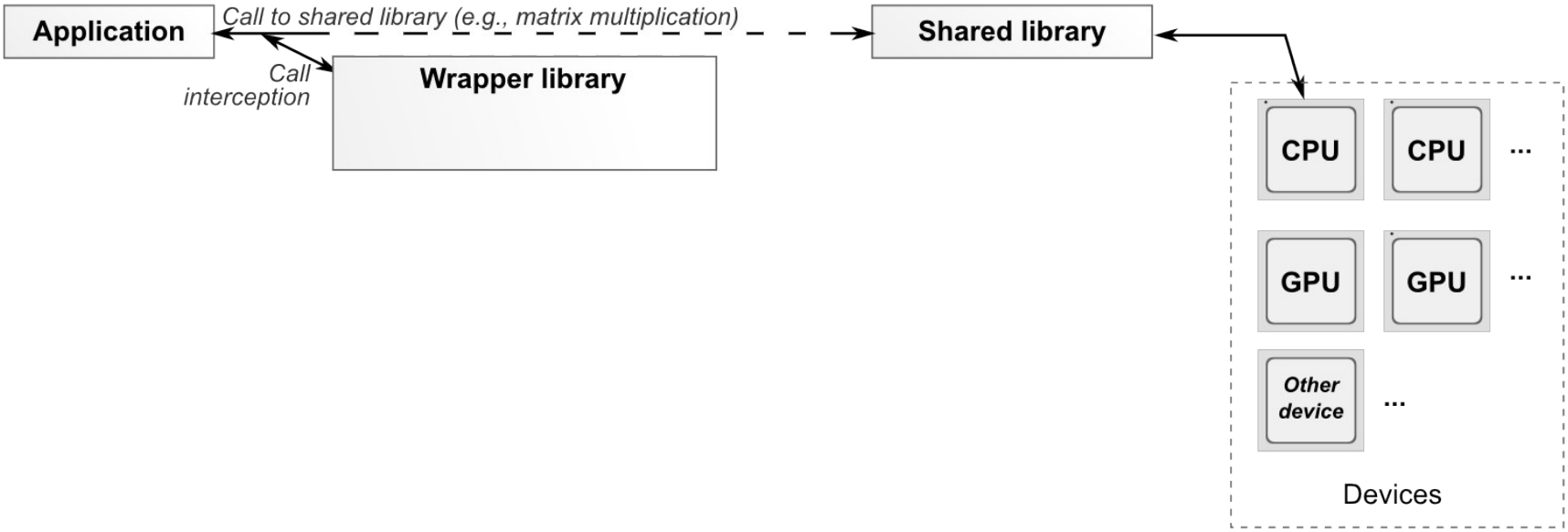
Framework Architecture

Framework Architecture



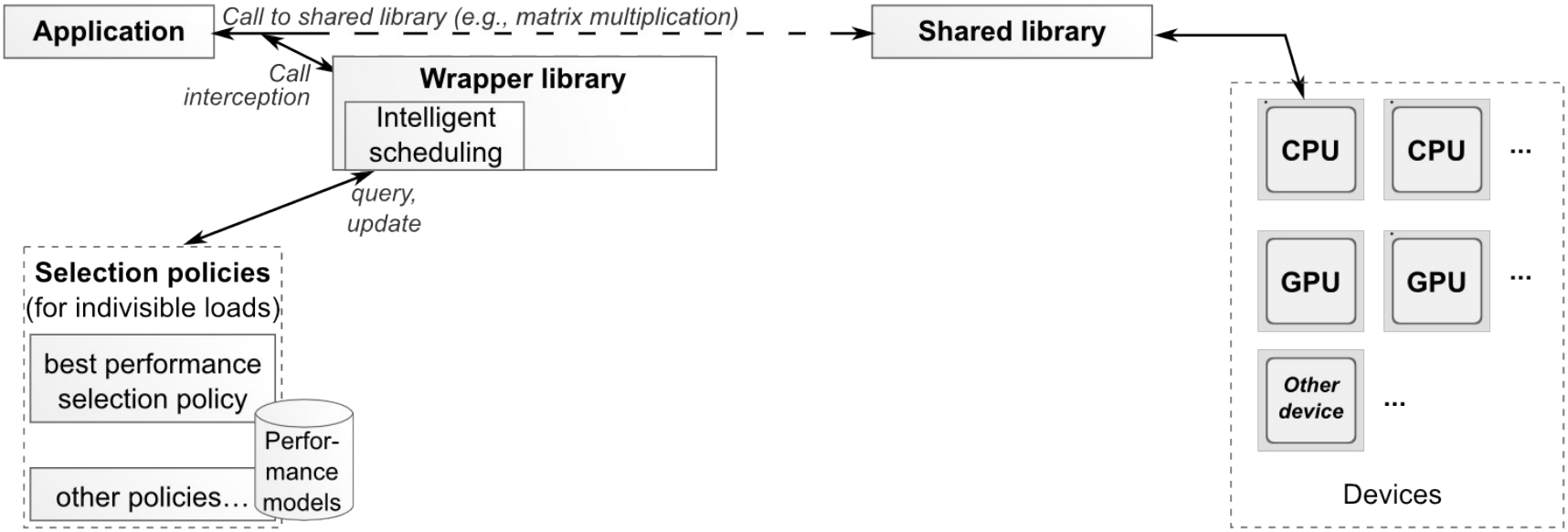
Legacy applications and libraries do not fully utilize modern heterogeneous computers

Framework Architecture



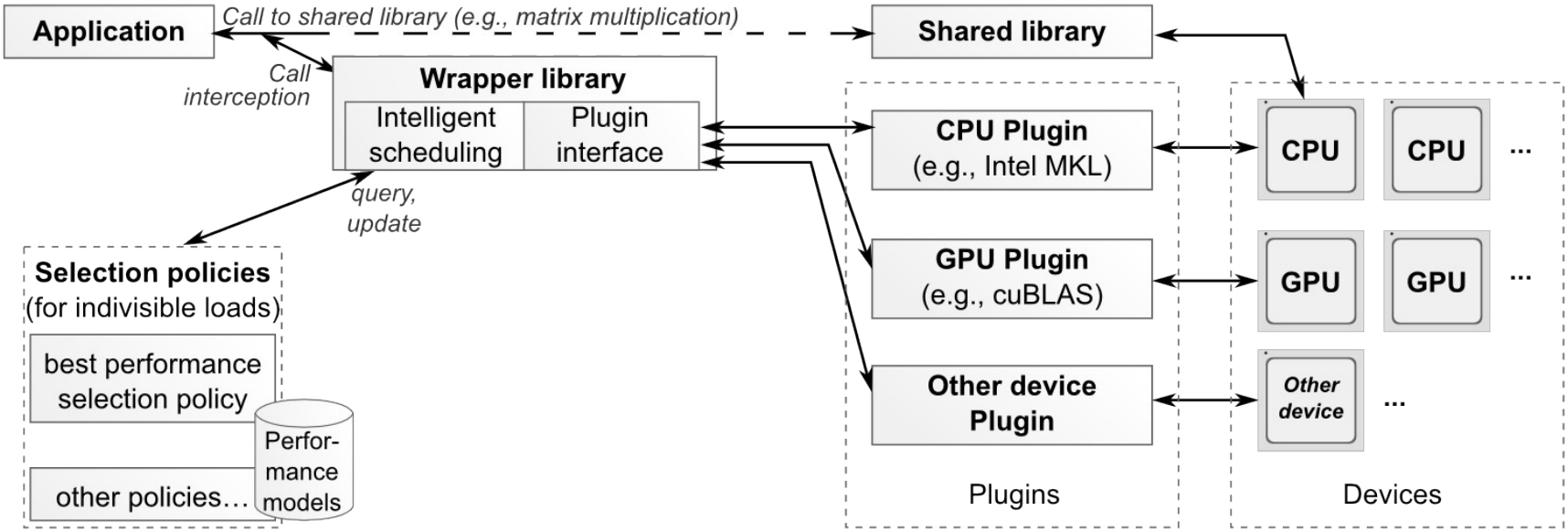
Calls to shared libraries can be intercepted...

Framework Architecture



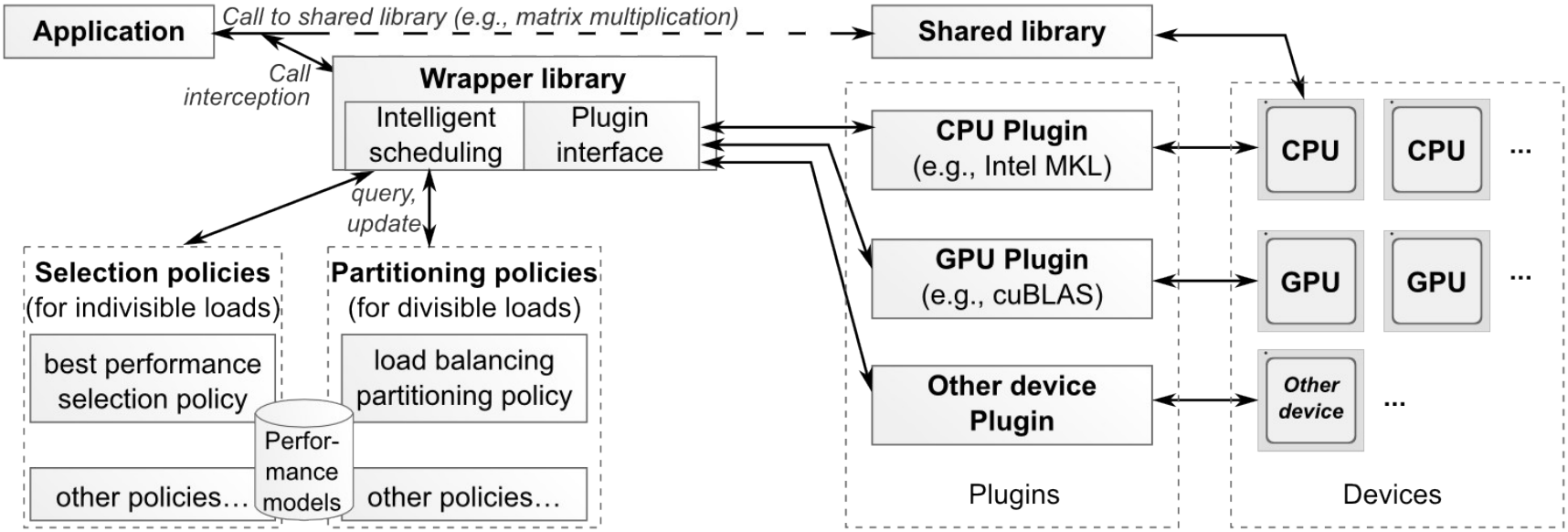
Calls to shared libraries can be intercepted to choose an optimized implementation for a particular call based on the problem size...

Framework Architecture



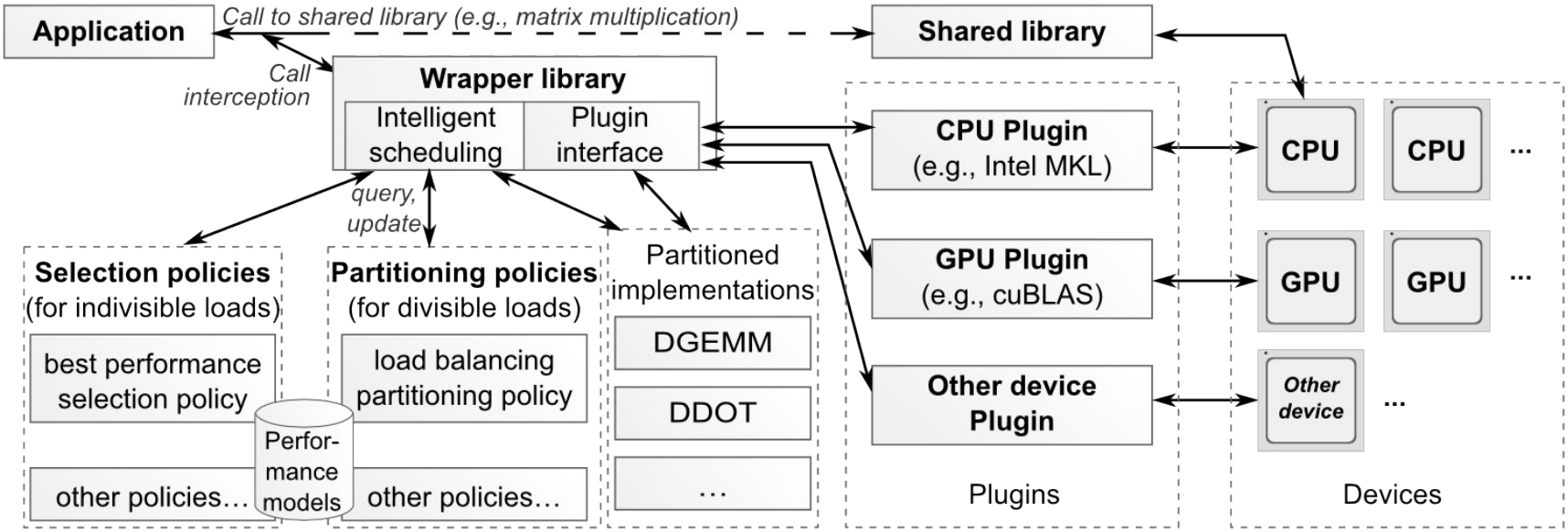
Calls to shared libraries can be intercepted to choose an optimized implementation for a particular call based on the problem size and execute it.

Framework Architecture



Some tasks can be partitioned so that multiple devices execute their portions simultaneously.

Framework Architecture



Partitioning depends on the particular function.

Outline

Framework Architecture

Selection and Partitioning policies

Library Generation

- **Run-time adaptive policies**
  - Best-performance selection
  - Load balancing

Experimental results

Accelerating BLAS and FFTW

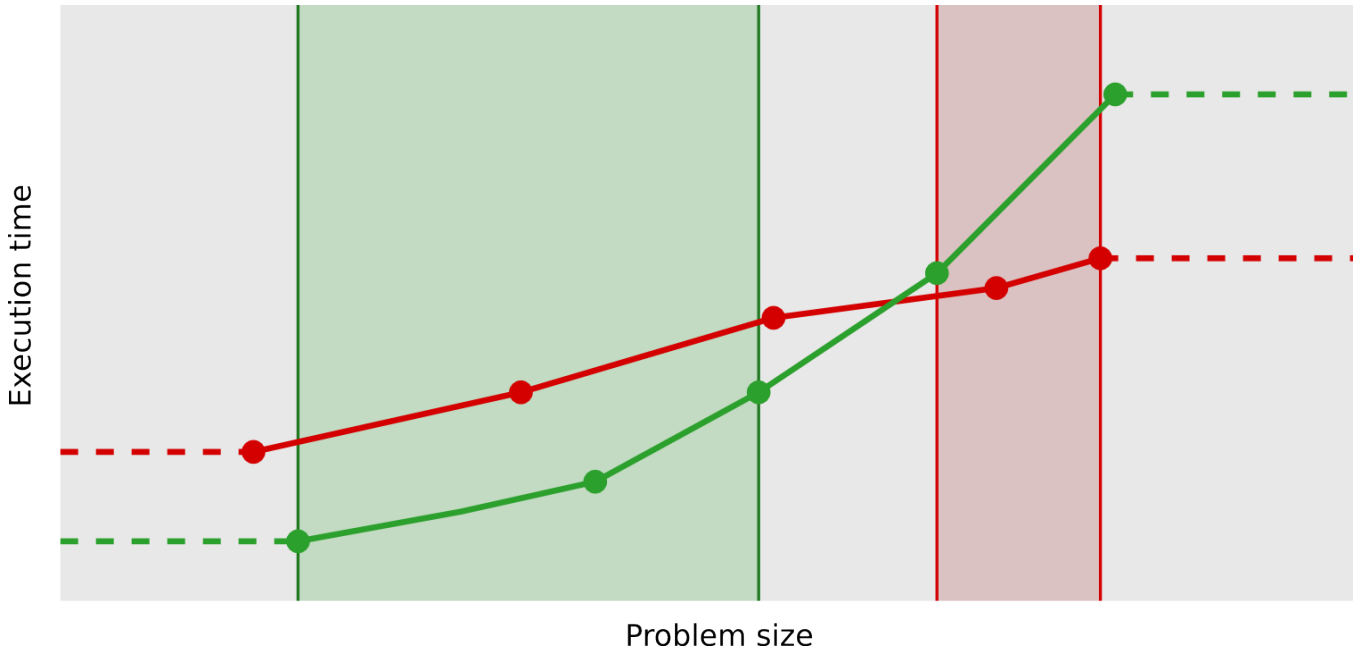
Overheads

Best performance selection policy (for indivisible workloads)

- Input – problem size
- Output – which *plugin* to run?

- Performance model – ordered set of pairs *(problem size, execution time)*
- For a given input problem size, find an exact match or the two neighboring points in the performance model.
- Which *plugin* offers the best performance?
- If cannot determine, run *both*.
- Update the model after the call finishes.

Best performance selection policy Example



Load balancing policy

- For workloads that can be efficiently partitioned among multiple devices.
- Used Lastovetsky and Reddy's algorithm (*Functional Performance Models*).
- The optimal overall execution time is when all devices finish at the same time.
- Input – problem size.
- Output – partial sizes of portions to be assigned to successive devices.
- If the model is empty – split equally.

Outline

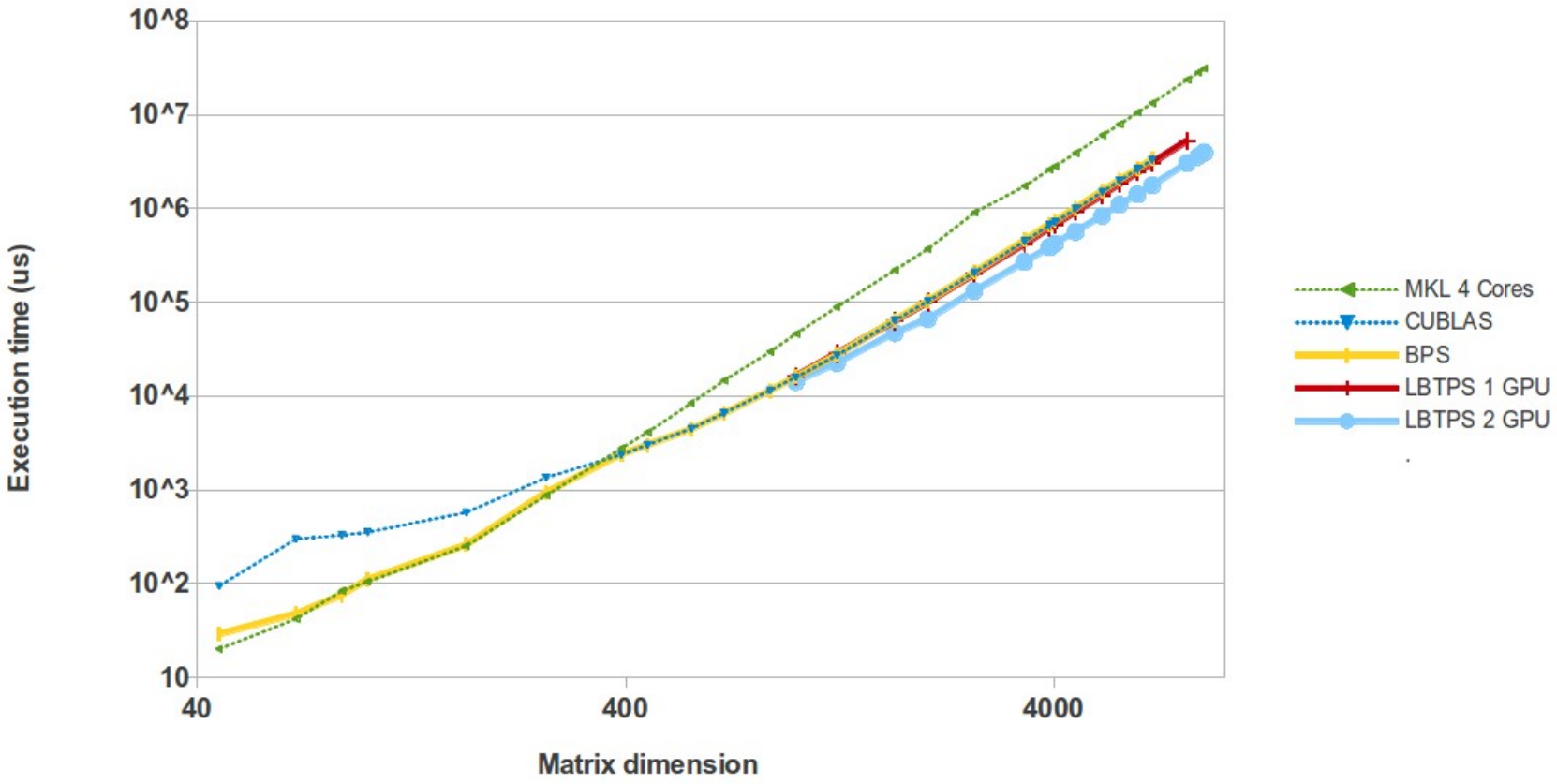
- **Framework Architecture**
  - Selection and Partitioning policies
  - Library Generation
- **Run-time adaptive policies**
  - Best-performance selection
  - Load balancing
- **Experimental results**
  - Accelerating BLAS and FFTW
  - Overheads

Experimental setup

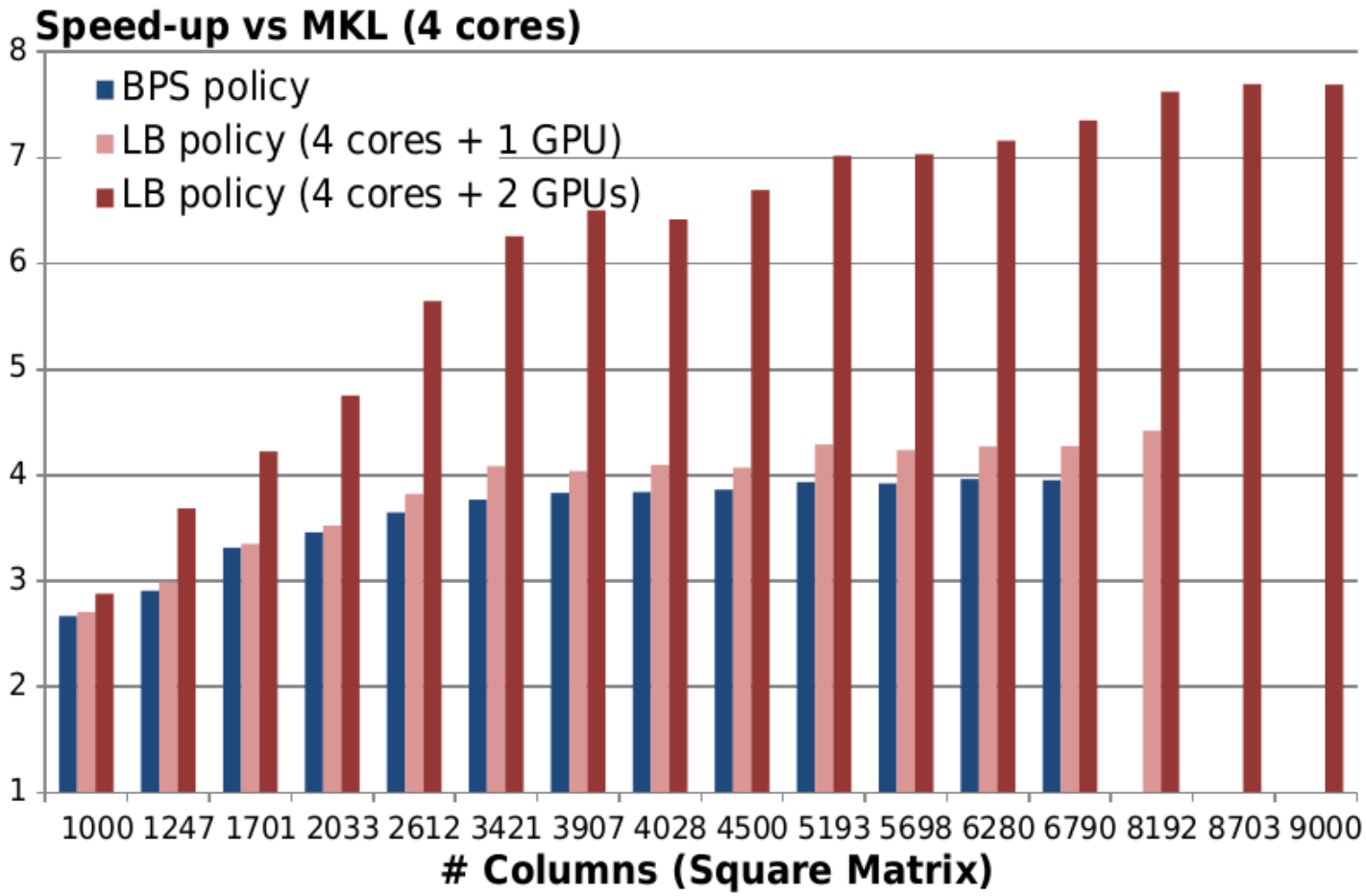
- CPU: Intel i7-950 (4 cores, 3.07 GHz)
- RAM: 12 GB DDR3-1033
- GPU: 2x NVIDIA GTX 580

- Reference implementations (CPU only):
  - MKL BLAS
  - FFTW3
- Application: Octave

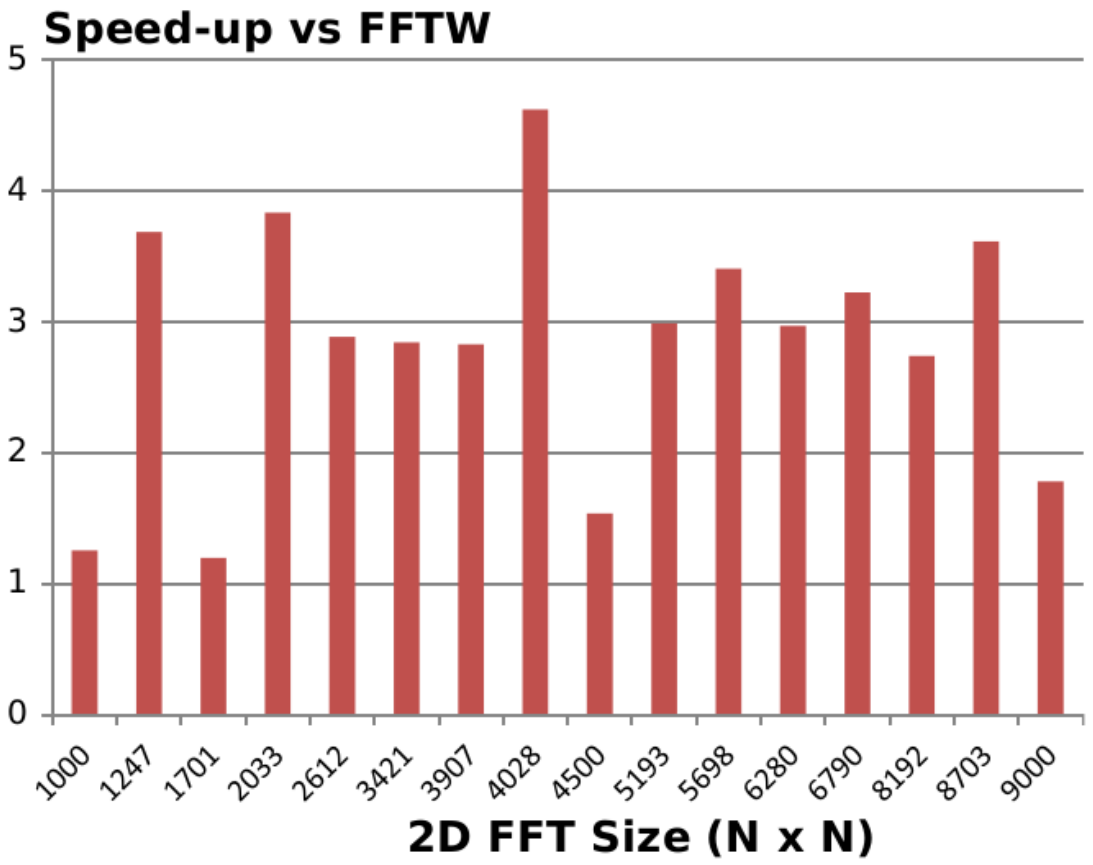
BLAS dgemm
Execution time



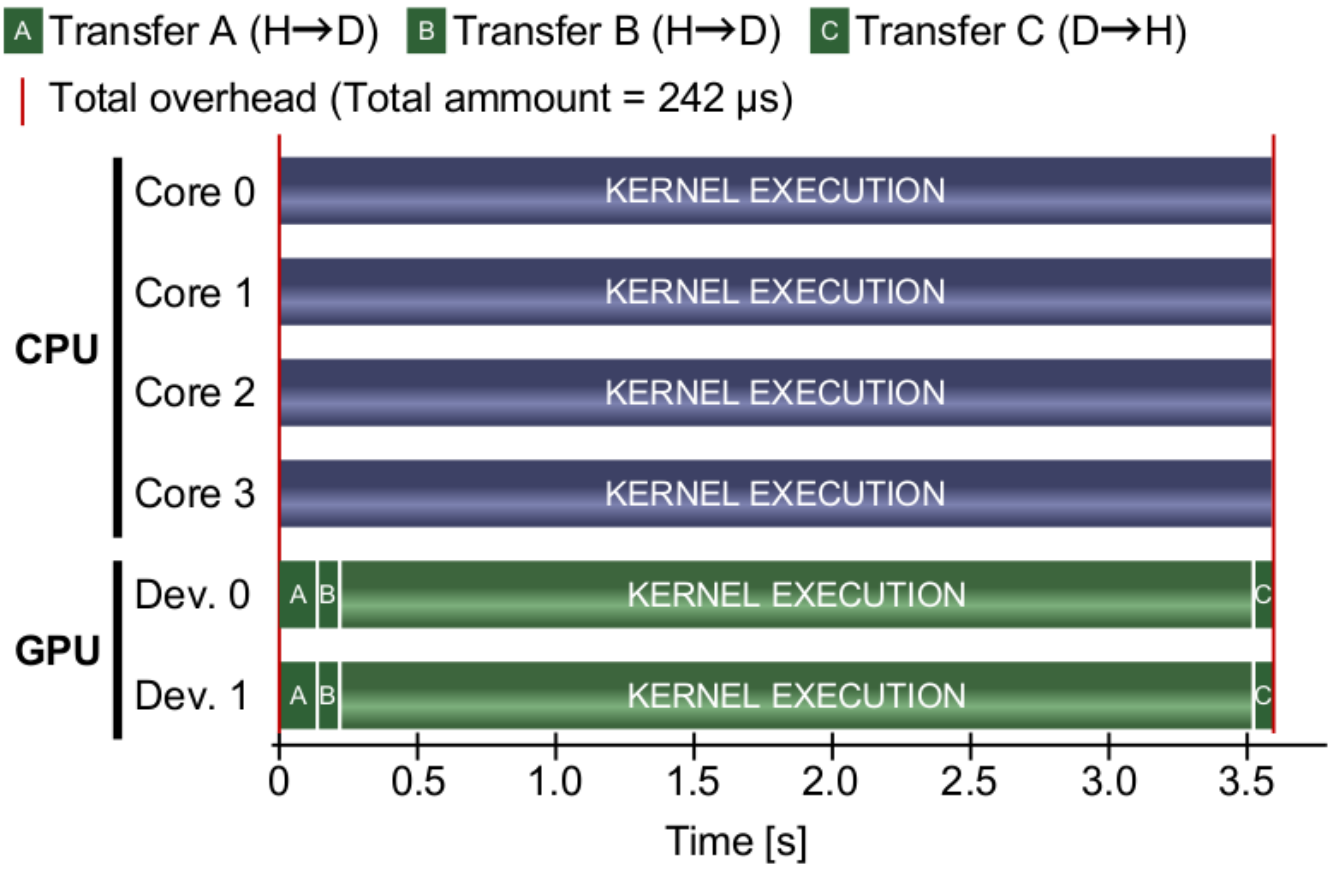
BLAS dgemm
Speedup over MKL with 4 cores



FFT 2D
Speedup over FFTW



Temporal diagram with load balancing



BLAS dgemm function, multiplying two 8703x8703 matrices.

Overheads

Source of overhead		#	Time
Library function interception, redirection and return		C	0.16 µs
BPS	Model update	1	0.34 µs
	Thread dispatch	D	36 µs
	Selection	C	3.16 µs
LB	Model update	C	0.42 µs
	Computing the distribution	C	25.05 µs
	Thread dispatch	C × D	36 µs
cuFFT initialization		1	1.3 s
cuBLAS initialization		1	0.273 s

C – number of calls for a given work size

D – number of devices

Summary

- **Transparently** accelerate existing applications without any modifications to them.
- Use **dynamic**, adaptive scheduling and partitioning policies.
- Speedup up to 7.86 (matrix multiplication) and 4.6 (FFT).

Thank you

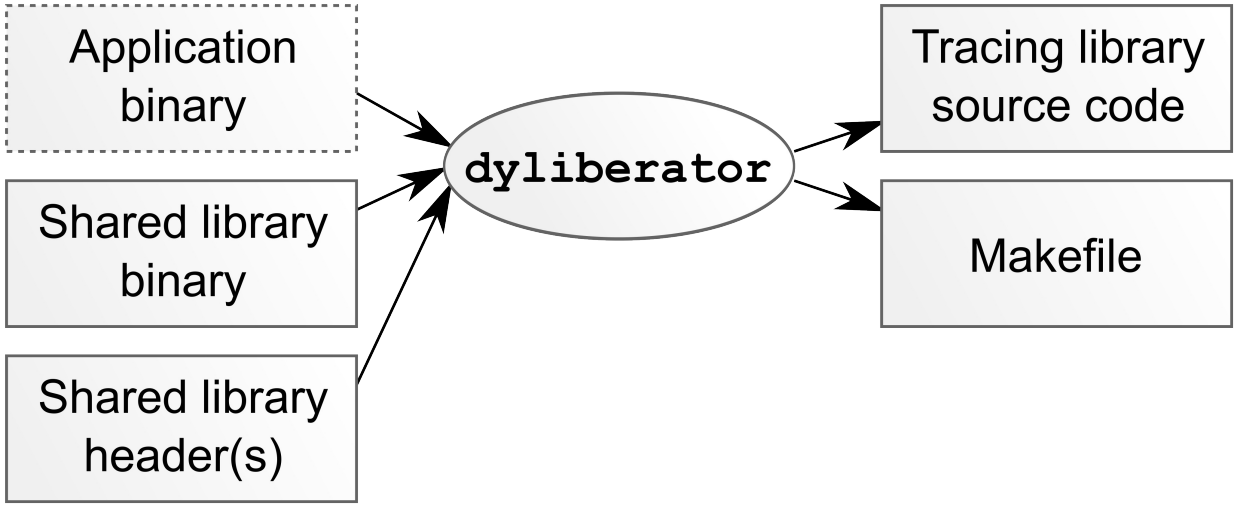
inesc id lisboa 10 anos

Questions?

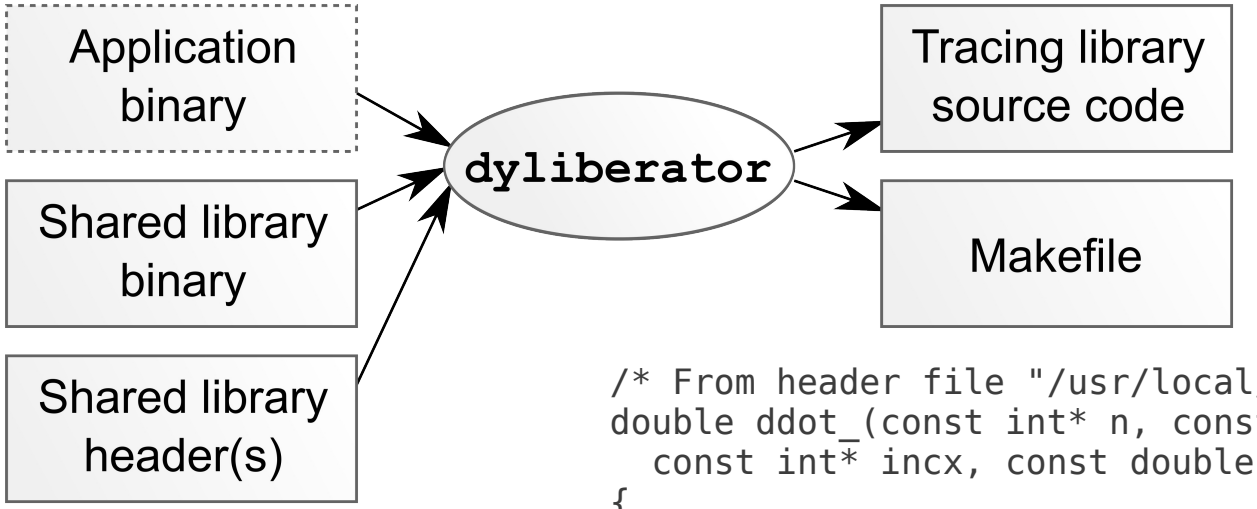
inesc id
lisboa
10 anos

Instituto de Engenharia de Sistemas e Computadores
Investigação e Desenvolvimento em Lisboa

Library generation



Library generation




```
/* From header file "/usr/local/include/blas.h" */
double ddot_(const int* n, const double* dx,
  const int* incx, const double* dy, const int* incy)
{
  double __result;
  /* generators.std.FuncPtr () */
  static double (*__funcPtr)(const int* n,
    const double* dx, const int* incx,
    const double* dy, const int* incy) = NULL;
  /* generators.std.Loader () */
  if (!__funcPtr)
    __funcPtr = __Dyl_loadFunction(__DYL_FID_ddot_);
  /* generators.std.Caller () */
  __result = (*__funcPtr)(n, dx, incx, dy, incy);
  return __result;
}
```